Technical report; TR-233

KNMI

Koninklijk Nederlands Meteorologisch Instituut

Vectorization of the ECBilt model

X. Wang and R.J. Haarsma

De Bilt, 2001

Technical Report = Technisch Rapport; TR-233

De Bilt, 2001

PO Box 201, 3730 AE De Bilt
The Netherlands
Wilhelminalaan 10
http://www.knmi.nl
Telephone +31 30 220 69 11
Telefax +31 30 221 04 07

Authors: X. Wang and R.J. Haarsma

UDC: 551.509.3
551.509.51
681.3.06

ISSN: 0169-1708

ISBN: 90-369-2192-9

Vectorization of the ECBilt Model

X. Wang and R.J. Haarsma

Technical report
KNMI
January 2001

1 Introduction

The ECBilt model (Haarsma et al, 1997, Opsteegh et al, 1998) code has been vectorized and tested on the Fujitsu VPP700 computer. In this report we describe the basic rules that we applied to vectorize the model, the vectorization procedures, performance gains and discussions on further improvements for the speedup of ECBilt. The ECBilt model is designed to run on the power-challenger which is a multi-processor scalar machine. When we transported this model onto the ECMWF's Fujitsu VPP700 we found that the performance of the model was rather disappointing. The CPU time for one year coupled run is 54 minutes which is a factor 2.25 more than the CPU time needed on the power-challenger. The percentage of vectorization is very low: 36%. This motivated us to optimize the code in order to make better use of the vector processor of the Fujitsu VPP700. After these optimizations the CPU time needed for one year coupled run is 5 minutes which is 5 times faster than on the power-challenger. The percentage of vectorization is 71%. Implementation of the vectorized code on the recently installed Fujitsu VPP5000 yielded another reduction in CPU time with a factor of 5.

2 About Vectorization

In this section some basic concepts about vectorization are given.

Vector operation: An operation on a set of elements of an array (or arrays), the result of which is independent of the ordering of the element operations. That is, the operation on any element of the array (or arrays) is independent of the result of the operation on any other element. Conceptually, we can think of the set of operations on all elements as happening simultaneously.

Vector registers: Like all modern computer processing units, the vector unit of the VPP cannot operate directly on operands in memory, the operands must be in the registers for faster access.

Vector length: Vector registers have a finite length (on the VPP this length is at most 2048) hence, in practice, VPP vector hardware instructions can only apply to length 2048 vectors. A *do* loop of length 10000 is actually executed as four vector instructions of length 2048 and one of 1808 length. Since the compiler concatenates these hardware vector length very efficiently, we refer to the *do* loop iteration as the vector length.

Vector length and performance

Vector length	Performance
1 to 8	A PC of Pentium variety performances better
8 to 25	A PC of SGI variety performances better
25 to 100	The VPP will overtake the SGI power-challenger
100 to 500	A worthwhile speedup can be expected
2048	Anything around here will be fast
greater than 10000	The VPP will outperform other machines

Vectorization:

(1) Compiler vectorization: A vectorizing compiler can be relied on to generate efficient vector instructions for most vectorizable code segments.

(2) User vectorization: Is the process of altering or adding information to the code to allow the compiler to generate more vector operations and overcome unnecessary bottlenecks in the programs execution.

3 General tuning principles

The principal aim in tuning a program for execution on the VPP is to minimize the total execution time for that program. This usually amounts to ensuring that a significant portion of the time-consuming code is executed in the vector unit of a processor of the VPP.

Although the compiler is good at generating vector instructions from standard FORTRAN code, there is no substitute for writing the code with the vector principles in mind if the code is to achieve the best performance.

A few basic rules are:

(1) Use *do* loops with as large iteration count as possible.

(2) Avoid complicated *if* and *goto* logic inside loops.

(3) Try to arrange inner *do* loop index as the first index of an array.

(4) Do as much arithmetic as possible within each loop for efficient use of the vector pipes.

(5) Be aware that load and store bottlenecks are often the limiting factors to high performance.

A loop cannot be vectorized when:

(1) The loop is too long.

(2) It contains function or subroutine calls.

(3) It contains *goto* statement.

(4) It contains *pause, return, stop* statements.

(5) It contains I/O with *end=* or *err=*.

(6) It contains *dowhile* and *dountil* loops.

Tuning a program for the VPP involves several stages. First, it is necessary to establish in which parts of the code the bulk of the CPU time is spent. Changes are then made in the time-consuming parts of the code to increase their rate of vectorization.

The process of identifying the time consuming parts of code and helping the compiler to vectorize them should be repeated several times, since, with significant improvements of the formerly time consuming sections of code, other parts may then start to dominate the computational costs.

4 Tuning and vectorizing the ECBilt model

Here we give a description of the major processes of optimizing the ECBilt code.

First of all we must identify the most expensive part of the model. This can be done by using the unix profiling tool. The compiling options used are: -Ds -CcdRR8 -X7 -w -Of -Wv,-Of,-m3 -Psa. In Table-1 the output from the profiling option is given. It gives the percentage of CPU time used by each routine relative to the total CPU time used by the whole model, vector length, percentage of the vector instructions relative to the total instructions of each routine and the routine's names respectively.

Percent	VL	V-Hit(%)	Name
54.8	10	41.6	**Lwaverad**
12.7	16	15.0	**Dlwrsfc**
5.3	12	4.2	**Tempprofile**
4.0	5	75.0	**Swaverad**
3.3	-	0.0	**Fluxsumland**
2.6	-	0.0	**Ptmoisgp**
1.7	-	0.0	**Detqmax**
1.7	-	0.0	**Qsat**
1.4	3	22.1	**Moisfields**
1.2	-	0.0	**Convec**
1.1	75	36.6	**Forward**
0.8	71	45.3	**Omega3**
0.7	-	0.0	**Zbrent**
0.7	-	0.0	**Albedo**
0.7	73	36.8	**Jacobd**
0.6	78	0.8	**Nanbks**
0.6	-	0.0	**Landtemp**
0.5	-	0.0	**Shine**
0.5	357	73.5	**C06fqu**
0.5	-	0.0	**Dragcoefgp**
0.5	144	1.8	**Sptogg**
...	...	...	...
...	...	...	...
...	...	...	...

Table-1

Below we will describe the vectorization of a number of the time consuming subroutines.

Lwaverad

From table-1 we see that the most time consuming routine is the **Lwaverad** routine. It takes more than one half of the total CPU time (which is 3240 seconds) and the percentage of vectorization is less than 50% (41.6%). The vector length is only 10.

The printout of this routine is in Appendix A1. In this and other appendices only the main body of each subroutine is listed.

Examining this routine we see the following main characteristics that prevent it from good vectorization.

(1) The double *do* loops are too long.

(2) It contains *write* statements in the double *do* loops.

(3) The inner loop contains the least number of iteration counts.

To modify it we need:

(1) Split the long *do* loops into a few shorter loops.

(2) Make a separate loop for the *write* statements. This loop is not vectorizable but it is a cheap loop.

(3) In order to split the long loop into shorter loops many variables need to contain grid information. This means to change single variables into multi-dimensional arrays. Variables with dimensions that contain no grid information should be extended to contain this information.

The modified version is given in Appendix A2. Because of the changes in this routine the total CPU time for one year coupled run dropped from 3240 to 1704 seconds. The vectorization percentage of it increased from 41.6 to 91.1%. It has now a vector length of 1619.

Tempprofile

This routine has a very low percentage of vectorization (4.2%) and a very short vector length. Looking at the code we see that it has the same problem as **Lwaverad**: too long loops; the inner loop contains a small number of iteration counts etc.. What makes it different from the **Lwaverad** routine is that it contains *dowhile* statements. The result of the condition in the *dowhile* statement depends on each grid point. This makes it difficult to be rewritten in general vectorizable *do* loops. After studying carefully what these two double *do* loops do and using physical knowledge of the model we found it is possible to replace the double *do* loops by a few general vectorizable *do* loops. After these changes[1] the CPU time used by **Tempprofile** routine dropped from 172.0 to 12.3 seconds. The vectorization rate increased from 4.2 to 69.6%. The vector length increased from 12 to 1166.

Moisfields

Subroutine Moisfields also has a low percentage of vectorization and a very short vector length. In Appendix A3 the routine is listed. Investigation of the code revealed that in the double *do* loops the subroutine **Ptmoisgp** (which again calls a function **Detqmax**) and the function **Qsat** are called. Both make it impossible to be vectorized. The simplest way of changing this is to inline subroutine **Ptmoisgp** and function **Qsat**. However inlining these subroutines directly makes the program messy. Inspection of the subroutine and the function reveals that many variables can be written in a statement function which is internal to the subroutine so that the routine can be vectorized. Those parts which are not easy to be written in a statement function are inlined here. In Appendix A3.1 the subroutine **Ptmoisgp** is given. In Appendix A3.2 the function **Detqmax** that is called by **Ptmoisgp** is listed. In Appendix A3.3 the function **Qsat** is listed. In Appendix A4 the modified version of **Moisfields** is given. In Appendix A4.1 the included file moist.h in which the statement function is written is listed. After these changes the subroutine **Moisfields** is 100% vectorized. The CPU time used by **Moisfields** for one year coupled run dropped from 45.4 to 1.8

seconds. The vector length changed from 3 to 512.

Convec

Convec subroutine is 0.0% vectorized and belongs to the top few expensive routines. This routine has the same characteristics as the previous ones namely that it contains loops that are too long, loops that contain subroutine calls and there are *write* statements in the loops. More over it contains *goto* statements in the loops to determine whether or not to stop with the convective adjustment. To vectorize these loops the *goto* statement should be certainly avoided. In the original version it allows a maximum of 10 times calls of convective adjustment. If more than 10 times is needed an error message is given and the model stops. We have checked the number of convective adjustments needed for each grid point on the earth and it turns out that the maximum number is 3. Therefore we apply 3 calls of convective adjustment for all the grid points so that the *goto* statement is not needed. The rest of the policy applied to modify this routine is the same as for the previous ones: split it into shorter loops (here it is splited into three subroutines); isolate the *write* statements in a separate loop; inline subroutines which are called in a *do* loop. Here the same subroutine **Ptmoisgp** is called as in **Moisfields**, so we can make use of the statement function moist.h. After these modifications the sum of the CPU time used by these 3 subroutines for one year coupled run is 5.1 seconds. The CPU time used by the **Convec** subroutine in the original model was 39.0 seconds. The vectorization rate of the **Convec1**, **Convec**2 and **Convec**3 is 99.1, 82.8 and 91.3% respectively. The vector length is 512, 398 and 1024 respectively.

Swaverad

Swaverad is the fourth most time consuming routine. Although it has a rather high vectorization percentage, the vector length is very short. Examining the code we see that it is characterized by long double *do* loops and short count of the inner loops. We can modify it by splitting it into shorter loops and make the outermost loop contain the least count of iterations. After these modifications the CPU time used by **Swaverad** routine dropped from 129.6 to 10.8 seconds, the vectorization percentage is increased from 75.0 to 82.8%. The vector length is increased from 5 to 1060.

Landtemp, Dlwrsfc, Fluxsumland, Zbrac and Zbrent

These are the surface temperature computing routines. In the model the surface temperature is computed from the assumption that the heat capacity of the surface is zero. This implies that the net heat flux between the atmosphere and the surface is zero. In the model an iteration procedure has been used to compute the balanced surface temperature for each grid point. How many iterations are needed for one grid point is different for each grid point. This procedure can not be vectorized. In order to speedup this procedure an alternative solution has to be sought. We decided to use another approach which assumes that the surface has a constant small heat capacity. The surface temperature can then be computed from an evolution equation which uses the net surface heat flux as input. In Appendix A5 the original subroutine **Landtemp** is listed. We see that in this routine in the double *do* loops subroutine **Zbrac** and function **Zbrent** are called which both use the external function **Fluxsumland**. **Fluxsumland** again calls a function **Dlwrsfc**. In Appendix A5.1 and A5.2 subroutine **Zbrac** and function **Zbrent** are listed respectively, while Appendix A5.3 and A5.4 contain the functions **Fluxsumland** and **Dlwrsfc** respectively. In Appendix A6 the modified version of the subroutine **Landtemp** is listed. The functions **Dlwrsfc**, **Fluxsumland** and **Zbrent** and subroutine **Zbrac** are not used any more. After this change the total CPU time needed for one year coupled run dropped from 984 to 350 seconds. The total percentage of vectorization of the model increased from 33.3 to 66.9%.

Albedo and **Shine**

Albedo and **Shine** are all 0.0% vectorized and in the top part of the listing of the most computational expensive subroutines. In routine **Albedo** a subroutine call to **Shine** is made in the double *do* loops which makes this subroutine impossible to be vectorized. One of the solutions for this is to rewrite these two subroutines into one. After these modifications the CPU time dropped from 38 to 1.2 seconds and vectorization rate is 97.8%.

Sptogg

So far the routines described all belong to the physics part of the model. In the dynamic part the **Sptogg** routine plays an important role. It is called many times by many other routines and has a very low vectorization rate (only 2.7%). In Appendix A7 and A8 the original version and modified version of routine **Sptogg** are given. With this modification the vectorization percentage of it increased from 2.7 to 97.4%. At the same time the vectorization percentage of the subroutines **Forward**, **Omega3** and **Jacobd** is also increased substantially. Due to the modification of this routine the total CPU time used for one year coupled run dropped from 320 to 300 seconds.

Final Result

The final result after the vectorization procedure is shown in Table-2. The most expensive subroutine is now **Forward** which has a vectorization rate of 78.7% and a vector length of 60. It consumes about 10% of the total CPU time. This is the most optimal result we could achieve. The total percentage of vectorization is 71%. The model is now 5 times faster on the VPP700 than on the power-challenger at KNMI.

5 Running the model on the Fujitsu VPP5000

As a test we have run the vectorized version of the atmosphere only model[2] on the ECMWF's Fujitsu VPP5000 machine which was put into use recently. The performance of the VPP5000 is drastically improved with respect to the VPP700. For one model year the CPU time needed is now only about one minute (70 seconds). While on the power-challenger at the KNMI the same run costs about 20 minutes. This is due to the fact that the VPP5000 processors have an increased peak performance and the scalar performance has significantly improved and also the performance on shorter vectors is better compared with the VPP700. In Table-3 the profiling of this run is given. From this table we see that it is very similar with the one given by the VPP700 with minor differences.

Percent	VL	V-Hit(%)	Name
10.3	60	78.7	**Forward**
8.3	44	93.2	**Omega3**
7.9	47	94.0	**Jacobd**
7.3	1619	91.1	**Lwaverad**
5.0	356	75.2	**C06fqu**
4.7	34	97.8	**Sptogg**
4.6	11	73.9	**Advec**
4.1	1166	69.6	**Tempprofile**
3.6	1060	82.8	**Swaverad**
3.4	36	93.4	**Psitogeo**
2.7	54	91.6	**Rggtosp**
...	...	...	...
...	...	...	...
1.2	512	99.1	**Convec1**
...	...	...	...
0.6	512	100.0	**Moisfields**
...	...	...	...
0.4	398	82.8	**Convec2**
...	...	...	...
0.4	617	97.1	**Albedo**
...	...	...	...
0.1	1024	91.3	**Convec3**
...	...	...	...

Table-2

6 Concluding Remarks

Till now we have reduced the CPU time of the model about 5 times on the VPP700 and 20 times on the VPP5000 compared with the original version running on the power-challenger at KNMI. The total percentage of vectorization is 71% on the VPP700 and 87% on the VPP5000. It seems difficult to improve on this. If we wish to get a even higher percentage, a major revision of the code has to take place. For instance, if all the multi-dimensional arrays are transferred into one-dimensional arrays this will speedup the model up to 30% or even more. The other point is that the advantage of the vector processor over the scalar processor will increase for increasing resolution. At present the horizontal resolution of the model is T21. We see in the final profile of Table-2 and Table-3 that although the most expensive dynamic routines are good vectorized the vector lengths are quite short. When the resolution is increased we expect that the vector length for these routines will increase, in this way we can make more efficient use of the vector processor (especially for the VPP700). Therefore if the resolution is increased the performance will increase compared to a scalar machine such as the power-challenger at KNMI.

Percent	VL	V-Hit(%)	Name
15.4	57	79.4	**Forward**
11.6	48	92.8	**Omega3**
10.7	55	92.7	**Jacobd**
10.1	1844	83.6	**Lwaverad**
8.0	32	96.7	**Sptogg**
5.9	720	66.2	**C06fqu**
4.3	35	91.1	**Psitogeo**
3.1	36	87.8	**Divwin**
3.0	1195	51.8	**Tempprofile**
2.7	32	95.1	**Rggtosp**
2.1	36	88.4	**Geowin**
1.8	-	0.0	**C06fpq**
1.7	32	94.8	**Ggtosp**
1.7	1061	70.6	**Swaverad**
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...

Table-3

References

Haarsma, R.J., F.M. Selten, J.D. Opsteegh, G. Lenderink and Q. Liu, 1997: ECBILT: A coupled atmosphere ocean sea-ice model for climate predictability studies. KNMI technical report TR-195, De Bilt, The Netherlands.

Opsteegh, J.D., R.J. Haarsma and F.M. Selten, 1998: ECBILT: A dynamic alternative to mixed boundary conditions in ocean models. Tellus, 50A, 348-367.

[1]For simplicity not all the routines are listed here. People who are interested please see CKO-website: http://www.knmi.nl/onderzk/CKO/ecbilt.html

[2]This model version is slightly different from the one mentioned in the previous sections.

Appendix A1

```
      subroutine lwaverad

      logco2=log(ghg(1)/ghgipcc(1))
      sqrch4=sqrt(ghg(2))-sqrt(ghgipcc(2))
      sqrn2o=sqrt(ghg(3))-sqrt(ghgipcc(3))


      is=imonth/3+1
      if (is.gt.4) is=1
      ism=(is-1)*3+1

      do i=1,27
        dqreg(i)=qancep(i,ism)**0.3333
      enddo

|-----do j=1,nlon
| |-----do i=1,nlat
| |         ireg(1)=irn(i,j,1)
| |         ireg(2)=irn(i,j,2)
| |         drmois=rmoisg(i,j)**0.3333
| |         if (rmoisg(i,j).lt.0.) then
| |           write(100,*) 'moisture less than zero'
| |           write(100,*) i,j,rmoisg(i,j),drmois
| |         endif
| |         dqa(1)=drmois-dqreg(ireg(1))
| |         dqa(2)=drmois-dqreg(ireg(2))
| |     |--do l=0,1
| |     |   |--do k=1,7
|c  *** loop| over sea (ireg(1)) or land regions (ireg(2))
| |     |   |  |--do nn=1,2
| |     |   |  |      lwrn(k,l,nn)=lwrref(k,ireg(nn),is,l)
| |   * |   |  |             +lwrqa(k,ireg(nn),is,l)*dqa(nn)
| |   * |   |  |             +lwrghg(k,1,ireg(nn),is,l)*logco2
| |   * |   |  |             +lwrghg(k,2,ireg(nn),is,l)*sqrch4
| |   * |   |  |             +lwrghg(k,3,ireg(nn),is,l)*sqrn2o
| |     |   |  |      do m=4,19
| |     |   |  |        lwrn(k,l,nn)=lwrn(k,l,nn)+
| |   * |   |  |           lwrghg(k,m,ireg(nn),is,l)*(ghg(m)-ghgipcc(m))
| |     |   |  |      enddo
| |     |   |  |      do m=1,ipl(ireg(nn))-1
| |     |   |  |        lwrn(k,l,nn)=lwrn(k,l,nn)+
| |   * |   |  |               lwrt(k,m,ireg(nn),is,l)*dtemp(m,i,j,nn)
| |     |   |  |      enddo
| |     |   |  |      lwrn(k,l,nn)=lwrn(k,l,nn)+
| |   * |   |  |         lwrt(k,18,ireg(nn),is,l)*dtemp(18,i,j,nn)
| |     |   |  |--enddo
| |     |   |--enddo
| |     |
| |     |      do k=1,7
| |     |        lwrnn(k,l,noc)=lwrn(k,l,l)
| |     |      enddo
| |     |
| |     |       dumts=tsurfn(i,j,noc)-tncep(19,ireg(1),ism)
| |     |  |--do m=1,4
| |     |  |      do k=1,3
| |     |  |        lwrnn(k,l,noc)=lwrnn(k,l,noc)+
| |   * |  |        (lwrts(k,m,ireg(1),is,l)+lwrqts(k,m,ireg(1),is,l)*dqa(1))
| |   * |  |        *dumts
| |     |  |      enddo
| |     |  |      lwrnn(7,l,noc)=lwrnn(7,l,noc)+
| |   * |  |      (lwrts(7,m,ireg(1),is,l)+lwrqts(7,m,ireg(1),is,l)*dqa(1))
| |   * |  |      *dumts
| |     |  |      dumts=dumts*(tsurfn(i,j,noc)-tncep(19,ireg(1),ism))
```

```
                |--enddo

                   do k=1,7
                     lwrnn(k,1,nse)=lwrn(k,1,1)
                   enddo

                   dumts=tsurfn(i,j,nse)-tncep(19,ireg(1),ism)
                |--do m=1,4
                      do k=1,3
                        lwrnn(k,1,nse)=lwrnn(k,1,nse)+
     *                 (lwrts(k,m,ireg(1),is,1)+lwrqts(k,m,ireg(1),is,1)*dqa(1))
     *                  *dumts
                      enddo
                      lwrnn(7,1,nse)=lwrnn(7,1,nse)+
     *                (lwrts(7,m,ireg(1),is,1)+lwrqts(7,m,ireg(1),is,1)*dqa(1))
     *                *dumts
                      dumts=dumts*(tsurfn(i,j,nse)-tncep(19,ireg(1),ism))
                |---enddo

                |--do k=1,7
                      lwrnn(k,1,nld)=lwrn(k,1,2)
                |--enddo

                    dumts=tsurfn(i,j,nld)-tncep(19,ireg(2),ism)
                |---do m=1,4
                       do k=1,3
                        lwrnn(k,1,nld)=lwrnn(k,1,nld)+
     *                 (lwrts(k,m,ireg(2),is,1)+lwrqts(k,m,ireg(2),is,1)*dqa(2))
     *                  *dumts
                      enddo
                      lwrnn(7,1,nld)=lwrnn(7,1,nld)+
     *                (lwrts(7,m,ireg(2),is,1)+lwrqts(7,m,ireg(2),is,1)*dqa(2))
     *                *dumts
                      dumts=dumts*(tsurfn(i,j,nld)-tncep(19,ireg(2),ism))
                |---- enddo

             |---enddo

c  ***  take weighted averages over ocean and land surfaces
            ulrad0(i,j)=0.0
            ulrad1(i,j)=0.0
            ulrad2(i,j)=0.0
            ulrads(i,j)=0.0
             dlrads(i,j)=0.0

             |--do nn=1,ntyps
                   ulrad0(i,j)= ulrad0(i,j) + fractn(i,j,nn)*
     *                 (lwrnn(1,0,nn)*(1-tcc(i,j))+lwrnn(1,1,nn)*tcc(i,j))
                   ulrad1(i,j)= ulrad1(i,j) + fractn(i,j,nn)*
     *                   ((lwrnn(2,0,nn)+lwrnn(5,0,nn))*(1-tcc(i,j)) +
     *                   (lwrnn(2,1,nn)+lwrnn(5,1,nn))*tcc(i,j))
                   ulrad2(i,j)= ulrad2(i,j) + fractn(i,j,nn)*
     *                   ((lwrnn(3,0,nn)+lwrnn(6,0,nn))*(1-tcc(i,j)) +
     *                   (lwrnn(3,1,nn)+lwrnn(6,1,nn))*tcc(i,j))
                   ulradsn(i,j,nn)=sboltz*tsurfn(i,j,nn)**4
                   dlradsn(i,j,nn)=-lwrnn(7,0,nn)*(1-tcc(i,j))-
     *                            lwrnn(7,1,nn)*tcc(i,j)
                   ulrads(i,j)=ulrads(i,j)
     *                        +fractn(i,j,nn)*ulradsn(i,j,nn)
                   dlrads(i,j)=dlrads(i,j)
     *                        +fractn(i,j,nn)*dlradsn(i,j,nn)
             |--enddo
            dumt2(i,j,2)=ulrad1(i,j)
            dumt1(i,j,1)=dlrads(i,j)
 12         dumt1(i,j,2)=ulrads(i,j)
```

```
|  |------enddo
|------enddo

      return
      end
```

Appendix A2

```
      subroutine lwaverad

      logco2=log(ghg(1)/ghgipcc(1))
      sqrch4=sqrt(ghg(2))-sqrt(ghgipcc(2))
      sqrn2o=sqrt(ghg(3))-sqrt(ghgipcc(3))


      is=imonth/3+1
      if (is.gt.4) is=1
      ism=(is-1)*3+1


      do i=1,27
        dqreg(i)=qancep(i,ism)**0.3333
      enddo

      do j=1,nlon
        do i=1,nlat
          ireg1=irn(i,j,1)
          ireg2=irn(i,j,2)
          dqa1(i,j)=rmoisg(i,j)**0.3333-dqreg(ireg1)
          dqa2(i,j)=rmoisg(i,j)**0.3333-dqreg(ireg2)
        enddo
      enddo

      do l=0,1
        do k=1,7
          do j=1,nlon
            do i=1,nlat
                xlwrn1(i,j,k,l)=xlwrref1(i,j,k,l,is)
     *                +xlwrqa1(i,j,k,l,is)*dqa1(i,j)
     *                +xlwrghg1(i,j,k,1,l,is)*logco2
     *                +xlwrghg1(i,j,k,2,l,is)*sqrch4
     *                +xlwrghg1(i,j,k,3,l,is)*sqrn2o


                xlwrn2(i,j,k,l)=xlwrref2(i,j,k,l,is)
     *                +xlwrqa2(i,j,k,l,is)*dqa2(i,j)
     *                +xlwrghg2(i,j,k,1,l,is)*logco2
     *                +xlwrghg2(i,j,k,2,l,is)*sqrch4
     *                +xlwrghg2(i,j,k,3,l,is)*sqrn2o

            enddo
          enddo
        enddo
      enddo

      do l=0,1
        do k=1,7
          do m=4,19
            do j=1,nlon
              do i=1,nlat
                xlwrn1(i,j,k,l)=xlwrn1(i,j,k,l)
     *                +xlwrghg1(i,j,k,m,l,is)*(ghg(m)-ghgipcc(m))

                xlwrn2(i,j,k,l)=xlwrn2(i,j,k,l)
     *                +xlwrghg2(i,j,k,m,l,is)*(ghg(m)-ghgipcc(m))
              enddo
            enddo
          enddo
        enddo
      enddo

      do l=0,1
```

```
        do k=1,7

         do m=1,17
          do j=1,nlon
            do i=1,nlat
              ireg1=irn(i,n,1)
              ireg2=irn(i,j,2)
                xlwrn1(i,j,k,l)=xlwrn1(i,j,k,l)
     *                +xlwrt1(i,j,k,m,l,is)*dtemp(m,i,j,1)
                xlwrn2(i,j,k,l)=xlwrn2(i,j,k,l)
     *                +xlwrt2(i,j,k,m,l,is)*dtemp(m,i,j,2)
              enddo
            enddo
          enddo

            do j=1,nlon
              do i=1,nlat
              xlwrn1(i,j,k,l)=xlwrn1(i,j,k,l)
     *                +xlwrt1(i,j,k,18,l,is)*dtemp(18,i,j,1)
              xlwrn2(i,j,k,l)=xlwrn2(i,j,k,l)
     *                +xlwrt2(i,j,k,18,l,is)*dtemp(18,i,j,2)
            enddo
          enddo

        enddo
      enddo

      do l=0,1
        do k=1,7
          do j=1,nlon
            do i=1,nlat
              xlwrnn(i,j,k,l,noc)=xlwrn1(i,j,k,l)
            enddo
          enddo
        enddo
      enddo

      do l=0,1
        do k=1,3
          do m=1,4
            do j=1,nlon
              do i=1,nlat
                dumts(i,j)=tsurfn(i,j,noc)-xtncep1(i,j,ism,19)
                xlwrnn(i,j,k,l,noc)=xlwrnn(i,j,k,l,noc)
     *             +(xlwrts1(i,j,k,m,l,is)+xlwrqts1(i,j,k,m,l,is)*dqa1(i,j))
     *             *dumts(i,j)**m
              enddo
            enddo
          enddo
        enddo

        do m=1,4
          do j=1,nlon
            do i=1,nlat
                dumts(i,j)=tsurfn(i,j,noc)-xtncep1(i,j,ism,19)
                xlwrnn(i,j,7,l,noc)=xlwrnn(i,j,7,l,noc)
     *             +(xlwrts1(i,j,7,m,l,is)+xlwrqts1(i,j,7,m,l,is)*dqa1(i,j))
     *             *dumts(i,j)**m
            enddo
          enddo
        enddo
      enddo
c      write(100,*)iday,xlwrnn(30,10,7,0,nse)

      do l=0,1
        do k=1,7
```

```
          do j=1,nlon
            do i=1,nlat
              xlwrnn(i,j,k,l,nse)=xlwrn1(i,j,k,l)
            enddo
          enddo
        enddo
      enddo

      do l=0,1
        do k=1,3
          do m=1,4
            do j=1,nlon
              do i=1,nlat
                dumts(i,j)=tsurfn(i,j,nse)-xtncep1(i,j,ism,19)
                xlwrnn(i,j,k,l,nse)=xlwrnn(i,j,k,l,nse)
     *            +(xlwrts1(i,j,k,m,l,is)+xlwrqts1(i,j,k,m,l,is)*dqa1(i,j))
     *            *dumts(i,j)**m
              enddo
            enddo
          enddo
        enddo
        do m=1,4
          do j=1,nlon
            do i=1,nlat
              xlwrnn(i,j,7,l,nse)=xlwrnn(i,j,7,l,nse)
     *             +(xlwrts1(i,j,7,m,l,is)+xlwrqts1(i,j,7,m,l,is)*dqa1(i,j))
     *             *dumts(i,j)**m
            enddo
          enddo
        enddo
      enddo
c       write(100,*)iday,xlwrnn(30,10,7,0,nse)
c       write(100,*)iday,xlwrnn(30,10,7,1,nse)

      do l=0,1
        do k=1,7
          do j=1,nlon
            do i=1,nlat
              xlwrnn(i,j,k,l,nld)=xlwrn2(i,j,k,l)
            enddo
          enddo
        enddo
      enddo

      do l=0,1
        do k=1,3
          do m=1,4
            do j=1,nlon
              do i=1,nlat
                dumts(i,j)=tsurfn(i,j,nld)-xtncep2(i,j,ism,19)
                xlwrnn(i,j,k,l,nld)=xlwrnn(i,j,k,l,nld)
     *            +(xlwrts2(i,j,k,m,l,is)+xlwrqts2(i,j,k,m,l,is)*dqa2(i,j))
     *            *dumts(i,j)**m
              enddo
            enddo
          enddo
        enddo
        do m=1,4
          do j=1,nlon
            do i=1,nlat
              dumts(i,j)=tsurfn(i,j,nld)-xtncep2(i,j,ism,19)
              xlwrnn(i,j,7,l,nld)=xlwrnn(i,j,7,l,nld)
     *             +(xlwrts2(i,j,7,m,l,is)+xlwrqts2(i,j,7,m,l,is)*dqa2(i,j))
     *             *dumts(i,j)**m
            enddo
          enddo
```

```
           enddo
         enddo

           do j=1,nlon
             do i=1,nlat
               ulrad0(i,j)=0.0
               ulrad1(i,j)=0.0
               ulrad2(i,j)=0.0
               ulrads(i,j)=0.0
               dlrads(i,j)=0.0
             enddo
           enddo

           do nn=1,ntyps
             do j=1,nlon
               do i=1,nlat

               ulrad0(i,j)= ulrad0(i,j) + fractn(i,j,nn)*
     *              (xlwrnn(i,j,1,0,nn)*(1-tcc(i,j))+xlwrnn(i,j,1,1,nn)*tcc(i,j))
               ulrad1(i,j)= ulrad1(i,j) + fractn(i,j,nn)*
     *                 ((xlwrnn(i,j,2,0,nn)+xlwrnn(i,j,5,0,nn))*(1-tcc(i,j)) +
     *                 (xlwrnn(i,j,2,1,nn)+xlwrnn(i,j,5,1,nn))*tcc(i,j))
               ulrad2(i,j)= ulrad2(i,j) + fractn(i,j,nn)*
     *                 ((xlwrnn(i,j,3,0,nn)+xlwrnn(i,j,6,0,nn))*(1-tcc(i,j)) +
     *                 (xlwrnn(i,j,3,1,nn)+xlwrnn(i,j,6,1,nn))*tcc(i,j))
               ulradsn(i,j,nn)=sboltz*tsurfn(i,j,nn)**4
               dlradsn(i,j,nn)=-xlwrnn(i,j,7,0,nn)*(1-tcc(i,j))-
     *                          xlwrnn(i,j,7,1,nn)*tcc(i,j)
               ulrads(i,j)=ulrads(i,j)
     *                        +fractn(i,j,nn)*ulradsn(i,j,nn)
               dlrads(i,j)=dlrads(i,j)
     *                        +fractn(i,j,nn)*dlradsn(i,j,nn)

             enddo
           enddo
         enddo

c         write(100,*)iday,dlradsn(30,10,nse)
         do j=1,nlon
           do i=1,nlat
             dumt2(i,j,2)=ulrad1(i,j)
             dumt1(i,j,1)=dlrads(i,j)
             dumt1(i,j,2)=ulrads(i,j)
           enddo
         enddo

         return
         end
```

Appendix A3

```
      subroutine moisfields
c-----------------------------------------------------------------------
c *** calculates relative humidity of the moised layer
c *** and specific humidity above the surface and at the surface
c-----------------------------------------------------------------------
      implicit none

      include 'comatm.h'
      include 'comphys.h'
      include 'comsurf.h'

      integer i,j,nn
      real*8  qsat,pmount,tmount,qmax,dqmdt

      do j=1,nlon
        do i=1,nlat

          call ptmoisgp(pmount,tmount,qmax,i,j,dqmdt)

          if (qmax.gt.0d0) then

            relhum(i,j)=min(1d0,rmoisg(i,j)/qmax)

          else

            relhum(i,j)=0d0

          endif

          q10(i,j)= 0.d0
          do nn=1,ntyps
            q10n(i,j,nn)=relhum(i,j) *
     *                 ec_qsat(pgroundn(i,j,nn),tempsgn(i,j,nn))
          enddo

c *** lwrmois is used in the lwr parameterization

          lwrmois(i,j)=rmoisg(i,j)**0.3333

        enddo
      enddo

      return
      end
```

Appendix A3.1

```
      subroutine ptmoisgp(pmount,tmount,qmax,i,j,dqmdt)

      z500=gpm500*grav
      hfac=2/rgas
      hred=hmoisr*grav
      pfac=log(plevel(2)/tlevel(2))

c *** calculate temperature at t500 assuming the temperature varies
c *** linearly with log(p) : T = Tref + alpha * log (p/pref)

      alpha=(temp2g(i,j) - temp4g(i,j))*rlogtl12
      t500 =temp4g(i,j) + alpha*pfac

c *** calculate reduced ground height in decameters
c *** reduction occurs in order to tune the amount of moisture which
c *** is allowed to pass a topographic barier

      hmount=qmount(i,j)*hred
      if (hmount.lt.0d0) hmount=0d0

c *** calculate the groundpressure assuming that the mean geopotential
c *** height at 500 hPa is gpm500 decameter
c *** calculate 10 mtr temperature in K

      tmount=t500**2 - hfac*alpha*(hmount-geopg(i,j,2)-z500)
      if (tmount.lt.0) then
        write(29,*) 'in latlon ',i,j
        write(29,*) tmount,hmount,t500,geopg(i,j,2)
        call ec_error(18)
      else
        tmount=sqrt(tmount)
      endif

c       pmount=plevel(2)*exp((tmount-t500)/alpha)

      qmax=ec_detqmax(tmount,i,j,dqmdt)

      return
      end
```

Appendix A3.2

```
      function detqmax(tmount,i,j,dqmdt)

      ti=temp4g(i,j)
      tj=tmount-temp4g(i,j)
      tk=temp4g(i,j)-temp2g(i,j)

      if (ti.lt.tqmi(0)) then
        ti=tqmi(0)
      endif
      if (ti.gt.tqmi(iqmtab)) then
        ti=tqmi(iqmtab)
      endif

      if (tj.lt.tqmj(0)) then
        tj=tqmj(0)
      endif
      if (tj.gt.tqmj(jqmtab)) then
        tj=tqmj(jqmtab)
      endif

      if (tk.lt.tqmk(0)) then
        tk=tqmk(0)
      endif
      if (tk.gt.tqmk(kqmtab)) then
        tk=tqmk(kqmtab)
      endif

      ii=min(iqmtab-1,int((ti-tqmimin)*rdtqmi))
      jj=min(jqmtab-1,int((tj-tqmjmin)*rdtqmj))
      kk=min(kqmtab-1,int((tk-tqmkmin)*rdtqmk))

      dqmdi=(qmtabel(ii+1,jj,kk)-qmtabel(ii,jj,kk))*rdtqmi
      dqmdj=(qmtabel(ii,jj+1,kk)-qmtabel(ii,jj,kk))*rdtqmj
      dqmdk=(qmtabel(ii,jj,kk+1)-qmtabel(ii,jj,kk))*rdtqmk

      qmax = qmtabel(ii,jj,kk) + (ti-tqmi(ii))*dqmdi +
     *      (tj-tqmj(jj))*dqmdj + (tk-tqmk(kk))*dqmdk
      if (qmax.lt.0d0) qmax=0d0

      if (qmax.gt.0.2) then
        write(29,*) 'in latlon ',i,j,' qmax ',qmax
        call ec_error(121)
      endif

      alpha=(temp2g(i,j)-temp4g(i,j))*rlogtl12
      t500=temp4g(i,j)+alpha*alogpl2tl2
      z500=gpm500*grav
      hmount=qmount(i,j)*hmoisr*grav

      dtgdt=(rgas*t500*alogtl1pl2 + (hmount-geopg(i,j,2)-z500))/
     *       (rgas*tmount*alogtl12)

      dqmdt=dqmdi + dqmdj * (dtgdt - 1d0) + dqmdk

      ec_detqmax=0.9*qmax

      end
```

Appendix A3.3

```
      function qsat(press,temp)
c-----------------------------------------------------------------------
c *** saturation mixing ratio
c *** input press in [Pa], temp in K
c *** output ec_qsat: saturation mixing ratio
c-----------------------------------------------------------------------
      implicit none
      include 'comatm.h'
      include 'comphys.h'

      real*8  press,temp,ec_qsat

      ec_qsat=cc1*exp(cc2*(temp-tzero)/(temp-cc3))
     &      /press

      end
```

Appendix A4

```
      subroutine moisfields
c-----------------------------------------------------------------------
c *** calculates relative humidity of the moised layer
c *** and specific humidity above the surface and at the surface
c-----------------------------------------------------------------------
      include 'moist.h'

      do j=1,nlon
        do i=1,nlat

          tmount=tmountx(i,j)

          ti=temp4g(i,j)
          tj=tmount-temp4g(i,j)
          tk=temp4g(i,j)-temp2g(i,j)
          ti=max(ti,tqmi(0))
          ti=min(ti,tqmi(iqmtab))
          tj=max(tj,tqmj(0))
          tj=min(tj,tqmj(jqmtab))
          tk=max(tk,tqmk(0))
          tk=min(tk,tqmk(kqmtab))

          ii=min(iqmtab-1,int((ti-tqmimin)*rdtqmi))
          jj=min(jqmtab-1,int((tj-tqmjmin)*rdtqmj))
          kk=min(kqmtab-1,int((tk-tqmkmin)*rdtqmk))

          dqmdi=(qmtabel(ii+1,jj,kk)-qmtabel(ii,jj,kk))*rdtqmi
          dqmdj=(qmtabel(ii,jj+1,kk)-qmtabel(ii,jj,kk))*rdtqmj
          dqmdk=(qmtabel(ii,jj,kk+1)-qmtabel(ii,jj,kk))*rdtqmk

          qmax = qmtabel(ii,jj,kk) + (ti-tqmi(ii))*dqmdi +
     *           (tj-tqmj(jj))*dqmdj + (tk-tqmk(kk))*dqmdk
          qmax=0.8*qmax
          if (qmax.lt.0d0) qmax=0d0

          if (qmax.gt.0d0) then
            relhum(i,j)=min(1d0,rmoisg(i,j)/qmax)
          else
            relhum(i,j)=0d0
          endif

          qsurf(i,j) = 0.d0
          q10(i,j)= 0.d0
        enddo
      enddo

      do nn=1,ntyps
        do j=1,nlon
          do i=1,nlat
            q10n(i,j,nn)=relhum(i,j) *
     &                   qsat(pgroundn(i,j,nn),tempsgn(i,j,nn))
            q10(i,j)=q10(i,j)+fractn(i,j,nn)*q10n(i,j,nn)
          enddo
        enddo
      enddo

      return
      end
```

Appendix A4.1

```
c***  moist.h:

      real*8 qsat
      real*8 zp,zt

      real*8  hmountx,hmounthelp
      real*8  alfa,tmountx,tmounthelp
      integer ipi,ipj

      real*8  t500_x,dtgdt,dqmaxdt,dqmdi,dqmdj,dqmdk


      qsat(zp,zt)=0.662*611.2*exp(17.67*(zt-273.15)/(zt-29.66))/zp

      alfa(ipi,ipj)=(temp2g(ipi,ipj) - temp4g(ipi,ipj))*rlogtl12

      hmounthelp(ipi,ipj)=qmount(ipi,ipj)*hmoisr*grav
      hmountx(ipi,ipj)=max(hmounthelp(ipi,ipj),0.0)

      tmounthelp(ipi,ipj)=(temp4g(ipi,ipj) +
     &                    alfa(ipi,ipj)*log(plevel(2)/tlevel(2)))**2 -
     &                    (2/rgas)*alfa(ipi,ipj)*
     &                    (hmountx(ipi,ipj)-geopg(ipi,ipj,2)-gpm500*grav)

      tmountx(ipi,ipj)=sqrt(tmounthelp(ipi,ipj))


      t500_x(ipi,ipj)=temp4g(ipi,ipj)+alfa(ipi,ipj)*alogpl2tl2

      dtgdt(ipi,ipj)=(rgas*t500_x(ipi,ipj)*alogtl1pl2 +
     &  (qmount(ipi,ipj)*hmoisr*grav-geopg(ipi,ipj,2)-gpm500*grav))/
     &      (rgas*tmountx(ipi,ipj)*alogtl12)

      dqmaxdt(ipi,ipj,dqmdi,dqmdj,dqmdk)=dqmdi+dqmdj*(dtgdt(ipi,ipj)-1d0)
     &                                  +dqmdk
```

Appendix A5

```
i      subroutine landtemp
c-----------------------------------------------------------------------
c ***  computes surface land temperature
c-----------------------------------------------------------------------
       external fluxsumland

c ***  tol is the wanted accuracy of the land temperature in degrees

       parameter (tol=0.1)

       common /landpoint/il,jl

       mitetel = 0

       do j=1,nlon
         do i=1,nlat
           dlandheat(i,j)=0d0
           nethfxland(i,j)=0d0
           if (fractn(i,j,nld).gt.epss) then

             il=i
             jl=j
             tsland=tland(il,jl)
             tsland1=tsland
             tsland2=tsland + 1.

             call zbrac(fluxsumland,tsland1,tsland2,itetel)
             if (itetel.eq.100) call error(7)
             tland(i,j)=zbrent(fluxsumland,tsland1,tsland2,tol,itetel)
             if (tland(i,j).lt.180.or.tland(i,j).gt.350) then
               write(100,*) 'tland out of range'
               write(100,*) i,j,tland(i,j)
             endif
             if (itetel.eq.100) call error(8)
             if (itetel.gt.mitetel) mitetel=itetel

             if (dsnow(i,j).gt.0d0.and.tland(i,j).gt.tzero) then
               tland(i,j)=tzero
               nethfxland(i,j)=fluxsumland(tzero)
             else
               dlandheat(i,j)=-fluxsumland(tland(i,j))
               nethfxland(i,j)=0d0
             endif
           endif
         enddo
       enddo

       return
       end
```

Appendix A5.1

```
      subroutine zbrac(func,x1,x2,iter)
c-----------------------------------------------------------------------
c *** this routine from numerical recipes determines an interval with
c *** bounds x1 and x2 that contains the root of the function func
c-----------------------------------------------------------------------

      implicit none

      integer   j,ntry,iter
      real*8    factor
      parameter (factor=1.6,ntry=100)
      real*8    f1,f2,x1,x2,func
      external  func

      f1=func(x1)
      f2=func(x2)
      do j=1,ntry
        if ((f1.le.0..and.f2.ge.0.).or.(f1.ge.0..and.f2.le.0.)) then
          iter=j
          return
        endif
        if (abs(f1).lt.abs(f2)) then
          x1=x1+factor*(x1-x2)
          f1=func(x1)
        else
          x2=x2+factor*(x2-x1)
          f2=func(x2)
        endif
      enddo
      return
      end
```

Appendix A5.2

```
      function zbrent(func,x1,x2,tol,iter)
c-----------------------------------------------------------------------
c *** this routine from numerical recipes determines the root of  the
c *** function func which is contained in the interval with bounds
c *** x1 and x2
c-----------------------------------------------------------------------
      a=x1
      b=x2
      fa=func(a)
      fb=func(b)
      if ((fa.gt.0.and.fb.gt.0.).or.(fa.lt.0.and.fb.lt.0.)) then
         call error(13)
      endif
      c=b
      fc=fb
      do iter=1,itmax
        if ((fb.gt.0.and.fc.gt.0.).or.(fb.lt.0.and.fc.lt.0.)) then
          c=a
          fc=fa
          d=b-a
          e=d
        endif
        if (abs(fc).lt.abs(fb)) then
          a=b
          b=c
          c=a
          fa=fb
          fb=fc
          fc=fa
        endif
        tol1=2.*eps*abs(b)+0.5*tol
        xm=0.5*(c-b)
        if (abs(xm).le.tol1.or.fb.eq.0.) then
          zbrent=b
          if (iter.gt.20) write(100,*) 'zbrent ',iter,b,fb
          return
        endif
        if (abs(e).ge.tol1.and.abs(fa).gt.abs(fb)) then
          s=fb/fa
          if (a.eq.c) then
            p=2.*xm*s
            q=1.-s
          else
            q=fa/fc
            r=fb/fc
            p=s*(2.*xm*q*(q-r)-(b-a)*(r-1.))
            q=(q-1.)*(r-1.)*(s-1.)
          endif
          if (p.gt.0.) q=-q
          p=abs(p)
          if (2.*p.lt.min(3.*xm*q-abs(tol1*q),abs(e*q))) then
            e=d
            d=p/q
          else
            d=xm
            e=d
          endif
        else
          d=xm
          e=d
        endif
        a=b
        fa=fb
```

```
     if (abs(d).gt.tol1) then
       b=b+d
     else
       b=b+sign(tol1,xm)
     endif
     fb=func(b)
     if (iter.gt.20) write(100,*) 'zbrent ',iter,b,fb
   enddo
   zbrent=b
   return
   end
```

Appendix A5.3

```
      function fluxsumland(tsland)
c-----------------------------------------------------------------------
c *** computes sum of fluxes between the land and the atmosphere
c-----------------------------------------------------------------------
c *** drag coefficient depends on landtemperature
      cdragl=dragcoefgp(tsland,il,jl,nld)

c *** sensible heatflux
      hfluxgp=alphad*cdragl*uv10(il,jl)*
     *                      (tsland-tempsgn(il,jl,nld))

c *** latent heat flux
      qsatss=qsat(pgroundn(il,jl,nld),tsland)

      if (dsnow(il,jl).gt.0d0) then
        edum=cdragl*uv10(il,jl)*(qsatss-q10n(il,jl,nld))
        edum=evfacan(il,jl,nld)*max(edum,0d0)
        esubf=alphas*edum
        evapf=alphav*edum
        esnow=min(rgridfac(il)*rowat*dsnow(il,jl)*rlatsub/dtime,
     &        esubf)
        if (esnow.lt.esubf) then
          sfrac=(esubf-esnow)/esubf
          emois=min(rgridfac(il)*rowat*bmoisg(il,jl)*rlatvap/dtime,
     &          sfrac*evapf)
          efluxgp=esnow+emois
        else
          efluxgp=esubf
        endif
      else
        efluxgp=alphav*cdragl*uv10(il,jl)*(qsatss-q10n(il,jl,nld))
        efluxgp=evfacan(il,jl,nld)*max(efluxgp,0d0)
        efluxgp=min(rgridfac(il)*bmoisg(il,jl)*rowat*rlatvap/dtime,
     &  efluxgp)
      endif

c *** downward longwave radiative flux

      dlradgp=dlwrsfc(il,jl,tsland,2)


c *** sum of all fluxes

      fluxsumland=heswsn(il,jl,nld) - hfluxgp + dlradgp -
     &            sboltz*tsland**4 - efluxgp + landheat(il,jl)


      return
      end
```

Appendix A5.4

```
      function dlwrsfc(i,j,tsland,irs)
c-----------------------------------------------------------------------
c *** computes downward long wave radiation at the land surface
c *** use to equilibrate land temperature
c *** output : dlrads(nlat,nlon): downward longwave radiation [Wm-2] at
c ***                             the surface
c-----------------------------------------------------------------------

      integer i,j,l,k,m,is,ism,ireg,irs
      real*8  lwr(7,0:1),logco2,sqrch4,sqrn2o,dqa,dumts,dlwrsfc,tsland

      logco2=log(ghg(1)/ghgipcc(1))
      sqrch4=sqrt(ghg(2))-sqrt(ghgipcc(2))
      sqrn2o=sqrt(ghg(3))-sqrt(ghgipcc(3))

      is=imonth/3+1
      if (is.gt.4) is=1
      ism=(is-1)*3+1

      ireg=irn(i,j,irs)
      dqa =(rmoisg(i,j))**0.3333-qancep(ireg,ism)**0.3333
      do l=0,1
        do k=7,7
          lwr(k,l)=lwrref(k,ireg,is,l)
     *                 +lwrqa(k,ireg,is,l)*dqa
     *                 +lwrghg(k,1,ireg,is,l)*logco2
     *                 +lwrghg(k,2,ireg,is,l)*sqrch4
     *                 +lwrghg(k,3,ireg,is,l)*sqrn2o
          do m=4,19
            lwr(k,l)=lwr(k,l)+
     *                    lwrghg(k,m,ireg,is,l)*(ghg(m)-ghgipcc(m))
          enddo
          do m=1,ipl(ireg)-1
            lwr(k,l)=lwr(k,l)+
     *                    lwrt(k,m,ireg,is,l)*dtemp(m,i,j,irs)
          enddo
          lwr(k,l)=lwr(k,l)+
     *                  lwrt(k,18,ireg,is,l)*dtemp(18,i,j,irs)
        enddo

        dumts=tsland-tncep(19,ireg,ism)
        do m=1,4
          lwr(7,l)=lwr(7,l)+
     *                  (lwrts(7,m,ireg,is,l)+lwrqts(7,m,ireg,is,l)*dqa)
     *                  *dumts
          dumts=dumts*(tsland-tncep(19,ireg,ism))
        enddo
      enddo
      dlwrsfc=-lwr(7,0)*(1-tcc(i,j))-lwr(7,1)*tcc(i,j)

      return
      end
```

Appendix A6

```
      subroutine landtemp
c-----------------------------------------------------------------------
c *** computes surface land temperature
c-----------------------------------------------------------------------

c** land heat capacity
      rhcapland=1.0/1500000.0
      hcapland=1500000.0


c** computes the sum of fluxes at the surface of the earth.
      do j=1,nlon
        do i=1,nlat
          fluxsumland(i,j)=(heswsn(i,j,nld)-hfluxn(i,j,nld)+dlradsn(i,j,nld)-
     &              ulradsn(i,j,nld)-efluxn(i,j,nld)+landheat(i,j))*gridfac(i)
        enddo
      enddo

      do j=1,nlon
        do i=1,nlat
          nethfxland(i,j)=0d0
          if (fractn(i,j,nld).gt.epss) then
            tland(i,j)=tland(i,j)+fluxsumland(i,j)*dtime*rhcapland
          endif
        enddo
      enddo



c *** in case of temperatures above zero in case of snowcover, set
c *** surface temperature to meltpoint

      do j=1,nlon
        do i=1,nlat
          if (fractn(i,j,nld).gt.epss) then
            if (dsnow(i,j).gt.0d0.and.tland(i,j).gt.tzero) then
                  nethfxland(i,j)=hcapland*(tland(i,j)-tzero)/dtime
                  tland(i,j)=tzero
            endif
          endif
        enddo
      enddo
      do j=1,nlon
        do i=1,nlat
          if (fractn(i,j,nld).gt.epss) then
            if (tland(i,j).lt.180.or.tland(i,j).gt.350) then
              write(100,*) 'tland out of range'
              write(100,*) i,j,tland(i,j)
            endif
          endif
        enddo
      enddo

      return
      end
```

Appendix A7

```fortran
      subroutine sptogg (as,agg,pploc)

c-----------------------------------------------------------------------
c *** conversion from spectral coefficients to gaussian grid
c *** input  spectral field as, legendre polynomials pploc (pp or pd)
c ***        where pp are legendre polynomials and pd derivatives with
c ***        respect to sin(fi)
c *** output gaussian grid agg
c-----------------------------------------------------------------------

      implicit none

      include 'comatm.h'
      include 'comdyn.h'

      integer i,ifail,j,k,k1,k2,m,mi,mr,nlon1
      real*8  as(nsh,2), agg(nlat,nlon), pploc(nlat,nsh)

c *** inverse legendre transform

      do j=1,nlon
        do i=1,nlat
          agg(i,j)=0.0d0
        enddo
      enddo

      nlon1=nlon+1
      k2=nshm(0)

      do k=1,k2
        do i=1,nlat
          agg(i,1)=agg(i,1)+as(k,1)*pploc(i,k)
        enddo
      enddo

      do m=1,nm
        mr=m+1
        mi=nlon1-m
        k1=k2+1
        k2=k2+nshm(m)
        do k=k1,k2
          do i=1,nlat
            agg(i,mr)=agg(i,mr)+as(k,1)*pploc(i,k)
          enddo
          do i=1,nlat
            agg(i,mi)=agg(i,mi)-as(k,2)*pploc(i,k)
          enddo
        enddo
      enddo

c *** inverse fourier transform

      ifail=0
      call c06fqf (nlat,nlon,agg,'r',trigi,wgg,ifail)

      return
      end
```

Appendix A8

```
      subroutine sptogg (as,agg,pploc)
c-----------------------------------------------------------------------
c *** conversion from spectral coefficients to gaussian grid
c *** input  spectral field as, legendre polynomials pploc (pp or pd)
c ***        where pp are legendre polynomials and pd derivatives with
c ***        respect to sin(fi)
c *** output gaussian grid agg
c-----------------------------------------------------------------------


      implicit none

      include 'comatm.h'
      include 'comdyn.h'

      integer i,ifail,j,k,k1,k2,m,mi,mr,nlon1
      real*8  as(nsh,2), agg(nlat,nlon), pploc(nlat,nsh)


c *** inverse legendre transform

      do j=1,nlon
        do i=1,nlat
          agg(i,j)=0.0d0
        enddo
      enddo

      nlon1=nlon+1
      k2=nshm(0)

      do k=1,k2
        do i=1,nlat
          agg(i,1)=agg(i,1)+as(k,1)*pploc(i,k)
        enddo
      enddo

      do j=1,231
        m=indexm(j)
        k=indexk(j)
        mr=m+1
        mi=nlon1-m
        do i=1,nlat
          agg(i,mr)=agg(i,mr)+as(k,1)*pploc(i,k)
          agg(i,mi)=agg(i,mi)-as(k,2)*pploc(i,k)
        enddo
      enddo

c *** inverse fourier transform

      ifail=0
      call c06fqf (nlat,nlon,agg,'r',trigi,wgg,ifail)

      return
      end
```

OVERZICHT VAN KNMI-PUBLICATIES, VERSCHENEN SEDERT 1999

KNMI-PUBLICATIE MET NUMMER

186-II Rainfall generator for the Rhine Basin: multi-site generation of weather variables by nearest-neighbour resampling / T. Brandsma a.o.
186-III Rainfall generator for the Rhine Basin: nearest-neighbour resampling of daily circulation indices and conditional generation of weather variables / Jules J. Beersma and T. Adri Buishand
186-IV Rainfall generator for the Rhine Basin: multi-site generation of weather variables for the entire drainage area / Rafal Wójcik, Jules J. Beersma and T. Adri Buishand
188 SODA workshop on chemical data assimilation: proceedings; 9-10 December 1998, KNMI, De Bilt, The Netherlands
189 Aardbevingen in Noord-Nederland in 1998: met overzichten over de periode 1986-1998 / [Afdeling SO]
190 Seismisch netwerk Noord-Nederland / [afdeling Seismologie]
191 Het KNMI-programma HISKLIM (HIStorisch KLIMaat) / T. Brandsma, F. Koek, H. Wallbrink, G. Können
192 Gang van zaken 1940-48 rond de 20.000 zoekgeraakte scheepsjournalen / Hendrik Wallbrink en Frits Koek
193 Science requirements document for OMI-EOS / contr. by R. van der A .. [et al.] **(limited distribution only)**

TECHNISCH RAPPORT = TECHNICAL REPORT (TR)

216 Evaluatierapport Automatisering Visuele Waarnemingen : Ontwikkeling Meestsystemen / Wiel Wauben e.a.
217 Verificatie TAF en TREND / Hans van Bruggen
218 LEO - LSG and ECBILT coupled through OASIS: description and manual / A. Sterl
219 De invloed van de grondwaterstand, wind, temperatuur en dauwpunt op de vorming van stralingsmist: een kwantitatieve benadering / Jan Terpstra
220 Back-up modellering van windmeetmasten op luchthavens / Ilja Smits
221 PV-mixing around the tropopause in an extratropical cyclone / M. Sigmond
222 NPK-TIG oefendag 16 december 1998 / G.T. Geertsema, H. van Dorp e.a.
223 Golfhoogteverwachtingen voor de Zuidelijke Noordzee: een korte vergelijking van het ECMWF-golfmodel (EPS en operationeel), de nautische gidsverwachting, Nedwam en meteoroloog / D.H.P. Vogelezang, C.J. Kok
224 HDFg library and some hdf utilities: an extention to the NCSA HDF library user's manual &reference guide / Han The
225 The Deelen Infrasound Array: on the detection and identification of infrasound / L.G. Evers and H.W. Haak
226 2D Variational Ambiguity Removal / J.C.W. de Vries and A.C.M. Stoffelen
227 Seismo-akoestische analyse van de explosies bij *S.E. Fireworks* ; Enschede 13 mei 2000 / L.G. Evers e.a.
228 Evaluation of modified soil parameterization in the ECMWF landsurface scheme / R.J.M. IJpelaar
229 Evaluation of humidity and temperature measurements of Vaisala's HMP243 plus PT100 with two reference psychrometers / E.M.J. Meijer
230 KNMI contribution to the European project WRINCLE: downscaling relationships for precipitation for several European sites / B.-R. Beckmann and T.A. Buishand
231 The Conveyor Belt in the OCCAM model: tracing water masses by a Lagrangian methodology / Trémeur Balbous and Sybren Drijfhout
232 Analysis of the Rijkoort-Weibull model / Ilja Smits
233 Vectorization of the ECBilt model / X. Wang and R.J. Haarsma

WETENSCHAPPELIJK RAPPORT = SCIENTIFIC REPORT (WR)

99-01 Enhancement of solar and ultraviolet surface irradiance under partial cloudy conditions / Serdal Tunç
99-02 Turbulent air flow over sea waves: simplified model for applications / V.N. Kudryavtsev et al.
99-03 The KNMI Garderen experiment, micro-meteorological observations 1988-89: corrections / Fred C. Bosveld
99-04 ASGAMAGE: the ASGASEX MAGE experiment : final report / ed. W.A.Oost
00-01 A model of wind transformation over water-land surfaces / V.N. Kudryavtsev et al.
00-02 On the air-sea coupling in the WAM wave model / D.F.Doortmont and V.K. Makin.
00-03 Salmon's Hamiltonian approach to balanced flow applied to a one-layer isentropic model of the atmosphere / W.T.M. Verkley
00-04 On the behaviour of a few popular verification scores in yes-no forecasting / C.J. Kok
01-01 Hail detection using single-polarization radar / Iwan Holleman
02-01 Comparison of modeled ozone distributions with ozonesonde observations in the tropics / Rob Puts