交通部中央氣象局委託研究計畫期末報告

中央氣象局全球預報模式新一代動力架構之發展與建置
Develop and Implement the New Generation Dynamics Core of CWB Global Forecast Model

計畫類別：□國內　☑國外
計畫編號：MOTC-CWB-98-3M-03
執行期間：98 年 2 月 23 日至 98 年 12 月 31 日
計畫主持人： 莊漢明
執行單位： Metsoft Corporation

中華民國　98　年 12　月

交通部中央氣象局九十八年度政府部門科技計畫期末摘要報告

計畫名稱：中央氣象局全球預報模式新一代動力架構之發展與建置

審議編號：		部會署原計畫編號：	MOTC-CWB-98-3M-03
主管機關：	交通部中央氣象局	執行單位：	Metsoft Corporation
計畫主持人：	莊漢明	聯絡人：	林美瑜
電話號碼：	1-301-299-2196	傳真號碼：	1-301-299-2196
期程：	98 年 2 月 23 日 至 98 年 12 月 31 日		
經費：（全程）	元	經費(年度)	788 仟元

執行情形：

1.執行進度：

	預定（％）	實際（％）	比較（％）
當年全程	100	100	0

2.經費支用：

	預定	實際	支用率（％）
當年全程	788 仟元	788 仟元	100

3.主要執行成果：[finish tracer semi-Lagrangian advection and compare to original one]

This project is to develop and implement non-iteration semi-Lagrangian tracer advection into CWB (Central Weather Bureau) GFS (Global Forecast System). Three major things have be implemented; (1) tracers are kept in grid-point space among input/integration/output, (2) transpose tracer in grid-point space in MPI computation are done, (3) mass conserving and positive definite interpolation in PPM are implemented. A case result of comparing with original Eulerian advection is given and indicates a successful implementation.

Detail final report is attached.

4.計畫變更說明：沒有變更

5.落後原因：沒有落後

6.主管機關之因應對策（檢討與建議）：

Final Report

Develop and Implement the New Generation Dynamics Core of CWB Global Forecast Model: semi-Lagrangian tracer advection

Hann-Ming Henry Juang

December 2,2009

Summary

For more than two decades, semi-Lagrangian transport method has been introduced, applied, and improved to have possibility of mass conserving and positive definition. It is applied to meteorological models; including regional, global, spectral and grid-point models. Thus, it is matured enough to implement into operational models. Based on the project requirement, we have implemented non-iteration dimensional splitting semi-Lagrangian advection for tracers with positive-definiteness in CWB GFS. The tracers in CWB GFS include specific humidity and cloud water.

In order to have mass conservation, tracer equation has to combine with continuity equation to form a mass conservation equation for tracers. In this case, it shows total integration over entire global is conserved while there is no source or sink for tracers. However, continuity equation has not to be implemented into this project yet, thus only tracer advection is implemented, nonetheless, mass-conserving positive-definite interpolation has been implemented into the CWB GFS. In the implementation, tracers are kept in grid-point space, and transpose grid points in spatial split computation for mass conserving and positive definite advection without halo in multi-processor cluster computing.

One case is selected to test this new modified code with semi-Lagrangain tracer advection in grid-point space, and to compare with original Eulerian tracer advection in spectral space. The wall time between them is about the same, it indicates that mass conserving positive-definite interpolation requires about the same computation as spectral transform. The results of 1 to 6hr integration from both methods are very similar, and they are starting to be different more after integration more in time.

1. Background for this project

It is well known that tracers in spectral model carry negative values if tracers are kept in spectral form in the model due to the spectral truncation from spectral transform. The spectral computation for tracers in spectral model is only the horizontal advection in Eulerian system, which requires horizontal derivative. If tracer-advection requires no horizontal derivative, then tracers can be kept in grid-point space, thus no negative values of tracers will be generated from spectral transform. One of the methods for advection without derivative is semi-Lagrangian advection.

Semi-Lagrangian advection is not new to meteorological society. It has been introduced by A. Robert et al (1985), and used for most of operational centers and research institutions for years, see the review paper by Staniforth and Cote (1991). Early problems of the semi-Lagrangian advection, such as non-conservation and possible negative values due to interpolation, and others, are one by one resolved. Several different methods are introduced, such as shape reserving, positive-definite, finite volume etc, to make semi-Lagrangian advection as a conserved and non-negative properties numerical method [such as Lin and Rood (1996), Zurroukat et al (2002) and (2005)].

Instead of following what others have done, we plan to implement recent implemented method, which is a non-iteration, spatial splitting, mass-conserving, and positive definite semi-Lagrangian advection to CWB GFS for tracers advection in this year's project (Juang 2007, 2008). Using mid-point in grid point, we don't need to guess and to do iteration to find the wind for advection. Spatial splitting can avoid the requirement for halo design in spectral model, which has no halo for spectral transform but grid transpose. Using transpose to have entire grid in any given direction can be easy to make mass conservation for the given direction. The non-negative interpolation is easy, either use monotonic PPM (piece-wise parabolic method) or monotonic Hermite interpolation. In this year, this semi-Lagrangian is applied to tracers only. Furthermore, this method has the same concept as finite volume to conserve mass but it has done without integral through the volume along the advection, thus it is considerable simpler.

2. Mass-conserving positive-definite semi-Lagrangan advection

The method of non-iteration, mass conserving, and positive definite semi-Lagrangian tracer advection has been published in the CWB conference proceeding in Juang 2007 and 2008. We will give finite differential equation in section a, then the descretization method in section b, as following.

a. Mass conserving tracer equations

For tracer equation without source and sink, it is simply written as

$$\frac{\partial q}{\partial t} = -m^2 u^* \frac{\partial q}{a\partial \lambda} - m^2 v^* \frac{\partial q}{a\partial \varphi} - \dot{\zeta}\frac{\partial q}{\partial \zeta}$$

where q is specific value of any given tracer, and the continuity or density equation in generalized vertical coordinates as shown in Juang 2005 can be written as

$$\frac{\partial(\partial p/\partial \zeta)}{\partial t} = -m^2\left(\frac{\partial u^*(\partial p/\partial \zeta)}{a\partial \lambda} + \frac{\partial v^*(\partial p/\partial \zeta)}{a\partial \varphi}\right) - \frac{\partial \dot{\zeta}(\partial p/\partial \zeta)}{\partial \zeta}$$

Combining above two equations, we can have flux form of tracer density equation as

$$\frac{\partial(\partial p/\partial \zeta)}{\partial t} = -m^2\left(\frac{\partial u^*(\partial p/\partial \zeta)}{a\partial \lambda} + \frac{\partial v^*(\partial p/\partial \zeta)}{a\partial \varphi}\right) - \frac{\partial \dot{\zeta}(\partial p/\partial \zeta)}{\partial \zeta}$$

Integral globally the above equation, we obtain

$$\iiint \frac{\partial}{\partial t}\left(\frac{\partial p}{\partial \zeta} q\right)\frac{a^2}{m^2} d\lambda d\varphi d\zeta = 0$$

It shows global mass conservation of any giving tracer. To maintain this conservation, any descretization for numerical model has to provide the same characteristic of this mass conservation. In the next section, we will show how to have a descretization to maintain this characteristic.

b. Mass conserving positive semi-Lagrangian advection

We will follow the method described in Juang (2007) to do semi-Lagrangian advection without iteration to find the mid-point wind. However, it started from advection form to do semi-Lagrangian, and spatial splitting. Since mass conserving can be easily obtained with flux form as the previous equation, let's start from the previous flux form equation but

consider advection form of some alternative, thus it can be written in one direction for example as

$$\frac{\partial A}{\partial t} = -\frac{\partial uA}{\partial x} = -u\frac{\partial A}{\partial x} - A\frac{\partial u}{\partial x}$$

where, for example,

$$A = \frac{\partial p}{\partial \zeta} q$$
$$x = \lambda$$

And the divergence term can be written as

$$\frac{\partial u}{\partial x} = \frac{1}{\Delta x}\frac{d\Delta x}{dt}$$

Then the flux form tracer density equation can be written in advection form with the consideration of mass conversation as following

$$\frac{dA\Delta x}{dt} = 0$$

so the final form for semi-Lagrangian equation become

$$(A\Delta x)^{n+1}_{arrival-point} = (A\Delta x)^{n-1}_{departure-point}$$

It tells us that mass (tracer density A times volume) is conserving in semi-Lagrangian advection. As long as we provide conditions during interpolation among following,

$$\sum_{global-sum}(A\Delta x)_{departure-point} = \sum_{global-sum}(A\Delta x)_{regular-point}$$

$$\sum_{global-sum}(A\Delta x)_{regular-point} = \sum_{global-sum}(A\Delta x)_{arrival-point}$$

then the total global tracer mass should be conserved. The interpolation method to have mass conservation, we can adopt monotonic piece-wise parabolic method (PPM) to construct conserving and positive interpolation with up to 3rd order to 4th order accuracy (see Appendix A). The edge of any given cell at departure, regular, and arrival condition is determined by

the wind speed at the edge of the middle time step from regular model cell. The global sum can be divided into latitudinal and longitudinal summation. In multi-processor parallel environment, grid-point transposition is used. First, all grids at the same latitude with some latitudes are transposed to compute advection along east and west direction, then transpose to have all grids at the same longitude with some longitudes as sub-domain to compute advection in north and south direction.

3. Implementation with routines modification and addition

Tables 1 and 2 list all routines either modified or newly created for this project. Appendix A illustrates the monotonic mass conserving interpolation for newly created code, and Appendix B lists all code comparison between original and current modified routines.

A logical variable called nislq is given as an option for integrating the model. It is given from Fortran namelist, and defined in const.h. If nislq is true, the model is running with mass-conserving positive-definite semi-Lagrangian for tracers in grid-point space without spectral transform. If nislq is false, the model is running with original Enlerian advection with spectral transform. Thus, most of the routines modified are related to the condition of nislq.

Table 1 lists all routines, which are modified to have option to run both nislq being true and false. Since the main differences between two options are (1) option to have tracer computation in spectral space or not and (2) option to have Eulerian or semi-Lagrangain advection. So, all routines related to spectral transform with tracers have to add option to do with or without tracer spectral transform (such as in getrdy.f, tendget.f, incrini.f, initial.f, and intgrt.f). All routines related to grid-point IO and spectral coefficient of tracer should be given with an option to have spectral coefficient or not, such as sigful.f, outflds.f, gather_spec.f, scatter_spec.f, with all spectral transform related routines as joinrs.f, joinsr.f, ujoinrs.f, ujoinsr.f). Routine used to do linear computation such as horizontal diffusion, hdiffu.f, should have no diffusion to tracer as well as routine for zonal implicit damping. The grid-point computation for advection has to have option to do Eulerian or semi-Lagrangian in main integration routine, intgrt.f, and mian forcing rotine, gridnl_hybrid.f.

Table 2 lists all new routines, which are specifically for non-iteration mass-conserving and positive-definite semi-Lagrangian tracer advection. The definition related to nislq is in nislq_def.f, the total 3D tracer advection computation is in nislq_advect.f, which has horizontal and vertical advections. The transpose from east-west to north-south and vice versa is in nislq_transpose.f, so that there is no halo in this implementation, even for grid-

point computation of horizontal advection. The completed package of non-iteration dimensional splitting mass-conserving and positive definite tracer advection is in nislq_pack.f, which is written as a module, including the cell definition of spherical Gaussian grid, horizontal and vertical monotonic interpolation with PPM, as illustrated in Appendix A.

4. Results

We have tested the newly developed model in two mode, original Eulerian tracer advection and semi-lagrangain tracer advection, and one arbitrary case has initial date on 1200 UTC 13 July 2007 with T240 with 30 layers in hybrid sigma-pressure coordinate. Two points are verified in the results. First point is to check the differences between two advections on tracers after short integration such as 6 hour and long integration such as 5 day alter. The second point is to check how the differences from tracer advection affect all other fields in prediction. From our experiences, we know both results from 6 hour should be very similar, but it should become different somewhat significantly after 5-day integration.

4.1 Comparison between Eulerian and semi-Lagrangian tracer advections on tracers

This subsection, we compare two kinds of results from original Eulerian advection and semi-Lagrangian tracer advection from Fig. 1 to Fig. 8. The original Eulerian advection requires spectral transform with notation as nislqF for nislq being false, and the implemented semi-Lagrangian advection doesn't require spectral transform, only in grid-point space, with notation as nislqT for nislq being true.

Figures 1 and 2 show relative humidity after 6 hr integration on 700 hPa and 100 hPa, respectively. They show their differences are not many except the south polar area due to spectral transform for Eulerian advection. Figures 3 and 4 are the same as Figs. 1 and 2 except after 24 hr integration, respectively. Figure 3 shows the same results as Fig. 1 that the wiggling patterns over south polar area are from Eulerian advection but not shown in semi-Lagrangian advection. This area is high terrain for southern pole region. Figure 4 shows more noise from Eulerian advection as compared to those from semi-Lagrangian advection at 100 hPa. And Figs 5 and 6 are the same results after 5-day integration. Since spectral Eulerian advection is high order scheme, the results from semi-Lagrangian look diffusive but reasonable and acceptable.

Figures 7 and 8 gave even clear pictures of their differences by using cloud water at 100 hPa. Since cloud water can be an isolated spotty area at high altitudes, such as 100 hPa. The spectral transform will show very clear artificial wave wiggling pattern all over the global, see Fig. 7 top panel. This kind of artificial wave would not be seen from semi-Lagrangian advection in bottom panel of Fig. 7 after 6 hr integration. This clear wiggling pattern will be

messed up after longer integration such as Fig. 8 top panel after 5-day integration, again not existed in semi-Lagrangian advection. This kind of artificial wiggling values is the reason to diverge the results very much from both runs after 5-day integration.

4.2 The influence to large-scale fields

From the previous sub-section, we realized that the spectral Gibbs effect and different computation of advection diverge the results more after integration longer. We would like to check the large-scale patterns, such as mean-sea-level pressure (MSLP) and 500 hPa height (Z500), even rainfall (PRCP) to examine their differences from Fig. 9 to Fig. 13.

Figures 9 shows the MSLP after 1-day integration. All the patterns are similar to each other between Eulerian and semi-Lagrangian advections. It is due to different order of numerical scheme, the storm over East Asian is weaker in semi-Lagrangian as compare to Eulerian scheme. After 5-day integration, MSLP are somewhat different shapes and patterns in some areas. Figures 11 shows an example of rainfall patterns, it indicates semi-Lagrangian tracer advection has the similar rainfall but a little bit diverged and/or diffusive.

Figures12 and 13 show the Z500 after 1- and 5-day integrations, respectively. We can see the large-scale patterns are very similar but, again, some diffusive in semi-Lagrangain scheme after 5-day integration.

5. Final discussion and concerns

From the results, though the third order interpolation in the scheme have some diffusive effects as compared to the original Eulerian scheme, the new semi-Lagrangian tracer advection gives positive definite tracer and indicates a reasonable and successful implementation. However, some questions and several concerns have to address for future improvements and further investigation.

The scores over large samples have to be established by conducting a quasi-operational parallel run to mark the performance of new semi-Lagrangian tracer advection. The performance or skill may have to fine tune through model physics and order of interpolation in semi-Lagragian scheme, though mass-conserving positive-definite semi-Lagrangian advection provides non-negative value of tracers for physics. Furthermore, even though semi-Lagrangian advection can have large time step, we have not tested it with different time step from the original code. More works have to be conducted later.

Reference

Junag, H.-M. H., 2007: Semi-Lagrangian advection without iteration. *Proceedings Conference on Weather Analysis and Forecasting, May 15-17, 2007, Acer Aspire Park, Longtan, Taoyuan*, p277.

____, 2008: Mass conserving and positive semi-Lagrangian tracer advection in NCEP GFS. *Proceedings Conference on Weather Analysis and Forecasting, September 9-11, 2008, Taipei*, p225-227.

Lin, S.-J. and R.B. Rood, 1996: Multidimensional flux-form semi-Lagrangian transport scheme. *Monthly Weather Review*, **124**, p2046-2070,

Robert, A. J., 1969: The integration of a spectral model of the atmosphere by the implicit method. Proceedings of the WMO/IUGG symposium on numerical weather prediction, Tokyo, Japan, November 26 – December 4, 1968, Japan Meteorological Agency, Tokyo, pp VII-19 – VII-24.

Staniforth, A. and J. Cote, 1991: Semi-Lagrangian schemes for atmosphere models – A review. *Monthly Weather Review*, **119**, p2206-2223.

Zerroukat, M., N. Wood, and A. Staniforth, 2002: SLICE: A semi-Lagrangian inherently conserving and efficient scheme for transport problems. Quart. J. Roy. Meteor. Soc., 128, 2801-2820.

____, ____, and ____, 2005: A monotonic and positive-definite filter for a semi-Lagrangian inherently conserving and efficient (SLICE) scheme. Quart. J. Roy. Meteor. Soc., 131, 2923-2936.

Table 1. Lists of modified routines

const.h	grid.h	
cons.f	gather_spec.f	getrdy.f
gridnl_hybrid.f	hdiffu.f	incrini.f
initial.f	intgrt.f	joinrs.f
joinsr.f	lgndr.f	matrix_hybrid.f
outflds.f	scatter_spec.f	sigful.f
tendget.f	ujoinrs.f	ujoinsr.f

Table 2. List of new routines

nislq_advect.f
nislq_def.f
nislq_pack.f
nislq_transpose.f
zimadv_nislq.f

These routines are in
/nwpr/gfs/xb48/henry/T240L30_hybrid_nodms/model_t240_64pe_q
under CWB xb48@61.60.103.123.

Appendix A

Monotonic Interpolation and Remapping

The scheme is mass conservation as long as the integration of the given profile within any given model layer equals the mean value times the length of the model grid cell. While the limiter is applied to the profile within the range of local given values, it is monotonic with positive definite interpolation. We provide PLM to have simple concept to understand before giving PPM, though PLM is not used in the implementation.

For the PLM (piece-wise linear method), a linear function passing through the mean value is used for all points within any given grid cell, the slope of the linear function is based on the values of the nearby grid cells, thus it is second order after integration, and it requires specific treatment to be monotonic. The following is a summary of the procedures for the PLM to be monotonic.

First, determine all slopes at all cell edges as

$$\hat{\delta}_i = (Q_i - Q_{i-1})/(x_i - x_{i-1}) \tag{A1}$$

where hat indicates a value at a cell edge, as mentioned, and $Q = \rho q$. Then the slope of any given cell is obtained by the mean of two nearby slopes at the cell edges. However, if the slopes at the two nearby cell edges have opposite signs, then the slope at the cell is reset to be zero. Thus the slope at any given cell k can be written as

$$\delta_i = \begin{cases} 0.5\left(\hat{\delta}_i + \hat{\delta}_{i-1}\right) & if \quad \hat{\delta}_i \hat{\delta}_{i-1} > 0 \\ 0.0 & otherwise \end{cases} \tag{A2}$$

We can then determine the values at cell edges for any given cell k as

$$\begin{aligned} Q_i^+ &= 0.5\delta_i\left(\hat{x}_{i+1} - \hat{x}_i\right) + Q_i \\ Q_i^- &= 2Q_i - Q_i^+ \end{aligned} \tag{A3}$$

where superscript + indicates the value at the cell edge of $\hat{x}_{i+1}$, and superscript – indicates the value at the cell edge of $\hat{x}_i$. The last check for a positive definition is to make sure there are non-negative values at cell edges, with the condition as

$$Q_i^+ = Q_i^- = Q_i \qquad if \quad Q_i^+ < 0 \;\; or \;\; Q_i^- < 0 \tag{A4}$$

So the profile for the integration of this linear function at given cell i can be written by values at cell edges as

$$Q = \left(Q_i^+ - Q_i^-\right)t + Q_i^- \qquad \text{(A5)}$$

where $t = (x - \hat{x}_i)/(\hat{x}_{i+1} - \hat{x}_i)$, which is between 0 and 1 from one edge to the other edge within a cell. Note that each cell has its own Q_i^+ and Q_i^-, thus Q may not be continuous at the cell edge, which is in a sense, piece-wise.

For the PPM (Colella and Woodward 1984), a parabolic function is used as the profile for any given point within the given grid cell. The parabolic function is based on values of the nearby grid cells as well, and it was a third order scheme after integration of the second order profile. The procedure to have monotonicity is as follows.

We followed "relaxed monotonicity for PPM" as proposed by Lin (2004) for our study with some corrections and modifications. First, we define the monotonic difference at any given cell k as

$$\Delta q_i^{mono} = sign\left(\min\left(abs\left(\Delta q_i\right), \Delta q_i^{\max}, \Delta q_i^{\min}\right), \Delta q_i\right) \qquad \text{(A6)}$$

where sign, max, min, and abs hereafter are the same usage as FORTRAN intrusive functions, and

$$\begin{aligned} \Delta q_i &= 0.25\left(Q_{i+1} - Q_{i-1}\right) \\ \Delta q_i^{\max} &= \max\left(Q_{i+1}, Q_i, Q_{i-1}\right) - Q_i \\ \Delta q_i^{\min} &= Q_i - \min\left(Q_{i+1}, Q_i, Q_{i-1}\right) \end{aligned} \qquad \text{(A7)}$$

Then, we determined the values at cell edges as

$$\begin{aligned} Q_i^- &= \frac{Q_{i-1}\left(\hat{x}_{i+1} - \hat{x}_i\right) + Q_i\left(\hat{x}_i - \hat{x}_{i-1}\right)}{\hat{x}_{i+1} - \hat{x}_{i-1}} - \frac{\Delta q_i^{mono} - \Delta q_{i-1}^{mono}}{3} \\ Q_{i-1}^+ &= Q_i^- \end{aligned} \qquad \text{(A8)}$$

For a positive definition, the following pass have to be performed on the values at cell edges as

$$\begin{aligned} Q_i^- &= Q_i - sign\left(\min\left(abs\left(2\Delta q_i^{mono}\right), abs\left(Q_i^- - Q_i\right)\right), \Delta q_i^{mono}\right) \\ Q_i^+ &= Q_i + sign\left(\min\left(abs\left(2\Delta q_i^{mono}\right), abs\left(Q_i^+ - Q_i\right)\right), \Delta q_i^{mono}\right) \end{aligned} \qquad \text{(A9)}$$

So the profile for integration of this parabolic function at given cell k can be written by the values at cell edges as

$$Q = 3\left(Q_i^+ + Q_i^- - Q_i\right)t^2 - 2\left(Q_i^+ + 2Q_i^- - 3Q_i\right)t + Q_i^- \qquad \text{(A10)}$$

where, again, $t = (x - \bar{x}_i)/(\bar{x}_{i+1} - \bar{x}_i)$ from 0 to 1 within any given cell. Again, each cell has its own Q_i^+ and Q_i^-, thus Q may not have been continuous at the cell edge, but is continuous inside any given cell, the so-called piece-wise method for monotonicity.

Appendix B
List of difference between original code and modified code

In this section, the comparison by diff between original code and modified code are summarized. The symbol < indicates modified code, symbol > indicates original code.

<Makefile >Makefile-orig

```
42c42
---
>               nislq_advect.f nislq_def.f nislq_pack.f nislq_transpose.f zimadv_nislq.f
85,86c84
---
>               nislq_advect.o nislq_def.o nislq_pack.o nislq_transpose.o zimadv_nislq.o
```

==========

< **lgndr.f** > **lgndr.f-orig**

```
1,2c1,2
< cfj   subroutine lgndr (myh,jtrun,mlmax,mlsort,sinl,poly,dpoly)
<       subroutine lgndr (myh,jtrun,jtmax,sinl,poly,dpoly)
---
> cfj   subroutine lgndr (my2,jtrun,mlmax,mlsort,sinl,poly,dpoly)
>       subroutine lgndr (my2,jtrun,jtmax,sinl,poly,dpoly)
9c9
< c  myh: number of gaussian latitudes from south pole and equator
---
> c  my2: number of gaussian latitudes from south pole and equator
27c27
<       dimension poly(jtrun,myh,jtmax),dpoly(jtrun,myh,jtmax),sinl(myh)
---
>       dimension poly(jtrun,my2,jtmax),dpoly(jtrun,my2,jtmax),sinl(my2)
38c38
<       do 1001 j=1,myh
---
>       do 1001 j=1,my2
```

==========

< **cons.f** > **cons.f-orig**

```
30c30
<      5              , tmeans,ptmeans,update,taureg,doincr,hybrid,nislq
---
>      5              , tmeans,ptmeans,update,taureg,doincr,hybrid
33c33
<       character*48 filist
---
>       character*60 filist
97,99c97
< chmhj idtg = 200000000000 + idtg8*100
<       idtg = 2.0e11
```

```
<       idtg = idtg         + idtg8*100
---
>       idtg = 200000000000 + idtg8*100
223c221
<       myh= my/2
---
>       my2= my/2
225c223
<       do 180 j = 1, myh
---
>       do 180 j = 1, my2
277,280c275
<       call lgndr (myh,jtrun,jtmax,sinl,poly,dpoly)
< !
< !  hmhj
<       if( nislq ) call nislq_init(nx,my,cosl,weight,nsize,myrank)
---
>       call lgndr (my2,jtrun,jtmax,sinl,poly,dpoly)

==========================================================
< matrix_hybrid.f   > matrix_hybrid.f-orig
112,113c112,113
< !     call mtxprt (dcdp,lev,1,'dcdp    ','f10.4   ')
< !     call mtxprt (dp,lev,1,'dp      ','f10.4   ')
---
>       call mtxprt (dcdp,lev,1,'dcdp    ','f10.4   ')
>       call mtxprt (dp,lev,1,'dp      ','f10.4   ')
165,168c165,168
< !     call mtxprt (arrhyd,lev,lev,'arrhyd  ','20f6.2  ')
< !     call mtxprt (eigval,lev,1,'eigval  ','f12.4   ')
< !     call mtxprt (evectr,lev,lev,'evectr  ','20f6.3  ')
< !     call mtxprt (evecin,lev,lev,'evecin  ','20f6.3  ')
---
>       call mtxprt (arrhyd,lev,lev,'arrhyd  ','20f6.2  ')
>       call mtxprt (eigval,lev,1,'eigval  ','f12.4   ')
>       call mtxprt (evectr,lev,lev,'evectr  ','20f6.3  ')
>       call mtxprt (evecin,lev,lev,'evecin  ','20f6.3  ')
175c175
< !     call mtxprt (pmcor,lev,1,'pmcor   ','f10.4   ')
---
>       call mtxprt (pmcor,lev,1,'pmcor   ','f10.4   ')
177c177
< !     call mtxprt (tmcor,lev,lev,'tmcor   ','20f6.3  ')
---
>       call mtxprt (tmcor,lev,lev,'tmcor   ','20f6.3  ')
```

```
==============================================================
< scatter_spec.f    > scatter_spec.f -orig
3c3
<      &  ,uzm,lev,ncld,jtrun,jtmax,my,nsize,nislq)
---
>      &  ,uzm,lev,ncld,jtrun,jtmax,my,nsize)
15d14
<       logical nislq
30,31d28
<
<       if( .not. nislq ) then
34,35d30
<       endif
<
44,45d38
<
<       if( .not. nislq ) then
48,49d40
<       endif
<
58,59d48
<
<       if( .not. nislq ) then
62,63d50
<       endif
<

==============================================================
< sigful.f    > sigful.f-orig
10c10
<      4       , hybrid, nislq )
---
>      4       , hybrid )
47c47
<       logical   hybrid, nislq
---
>       logical   hybrid
144,150c144,148
<       do jj = 1, jlistnum
<         j=jlist1(jj)
<           do i = 1, nx
<             shtmp(i,k,jj) = hld1(i,j)
<           enddo
<       enddo
<    73 continue
---
```

```
>       do 73 jj = 1, jlistnum
>        j=jlist1(jj)
>       do 73 i = 1, nx
>       sht(i,k,jj) = hld1(i,j)
>   73  continue
152c150
<       if( hybrid ) call convert_sig_hybrid(shtmp,pltinp,plt,
---
>       if( hybrid ) call convert_sig_hybrid(sht,pltinp,plt,
154,167d151
<
<       if( nislq ) then
<       do j=1,jlistnum
<         do k=1,lev
<           do i=1,nx
<             shtmp(i,k,j) = max(0.0,shtmp(i,k,j))
<           enddo
<         enddo
<       enddo
< !     call check_positive(shtmp,nx,lev,jlistnum,' q after convert_sig_hybrid ')
< !     call check_maxmin  (shtmp,nx,lev,jlistnum,' q after convert_sig_hybrid ')
<       endif
<
<       sht(:,1:lev,:) = shtmp(:,1:lev,:)
179,182c163,166
<         j=jlist1(jj)
<         do i = 1, nx
<           shtmp(i,k,jj) = hld1(i,j)
<         end do
---
>       j=jlist1(jj)
>       do i = 1, nx
> !hmhj sht(i,kk,jj) = hld1(i,j)
>       shtmp(i,k,jj) = hld1(i,j)
184a169
>       end do
188,199d172
<       if( nislq ) then
<       do j=1,jlistnum
<         do k=1,lev
<           do i=1,nx<             shtmp(i,k,j) = max(0.0,shtmp(i,k,j))
<           enddo
<         enddo
<       enddo
< !     call check_positive(shtmp,nx,lev,jlistnum,' cld after convert_sig_hybrid ')
< !     call check_maxmin  (shtmp,nx,lev,jlistnum,' cld after convert_sig_hybrid ')
```

```
<       endif
<
201d173
<
347c319
<       call joinrs(cc,ut,vt,dummy,dummy,nx,my_max,lev,jlistnum,2,0)
---
>       call joinrs(cc,ut,vt,dummy,dummy,nx,my_max,lev,jlistnum,2,1)
351c323
<      &          ,mlistnum,2,0)
---
>      &          ,mlistnum,2,1)
353d324
<       if( .not. nislq ) then
363,365d333
<       else
<            qrefs = 0.0
<       endif
498,533d465
<       return
<       end
<
<       subroutine check_positive( a, im, km, jm , ch )
<       character*(*) ch
<       real a(im,km,jm)
<       integer im,jm,km,i,j,k,numa,numq
<       numa=im*km*jm
<       numq=0
<       do i=1,im
<         do j=1,jm
<           do k=1,km
<             if( a(i,k,j).lt.0.0 ) numq=numq+1
<           enddo
<         enddo
<       enddo
<       print *,'negtive #',numq,'  of ',numa,' for ',ch
<       return
<       end
<
<       subroutine check_maxmin( a, im, km, jm , ch )
<       character*(*) ch
<       real a(im,km,jm)
<       integer im,jm,km,i,j,k
<       real qmax,qmin
<       qmax=a(1,1,1)
<       qmin=a(1,1,1)
```

```
<       do i=1,im
<         do j=1,jm
<           do k=1,km
<             qmax = max(qmax,a(i,k,j))
<             qmin = min(qmin,a(i,k,j))
<           enddo
<         enddo
<       enddo
<       print *,' max min ',qmax,qmin,' for ',ch


==========================================================
< tendget.f    > tendget.f-orig
43d42
<         if( .not. nislq ) then
47,49d45
<         else
<           qbar = 0.0
<         endif
76d71
<       if( .not. nislq ) then
83,85d77
<       else
<       qten = 0.0
<       endif
104,105c96,97
<         call gridnl_hybrid (nx,lev,ncld,nislq
<      *  , cp,radsq,ut(1,1,jj),vt(1,1,jj),pdot(1,1,jj),rdiv(1,1,jj)
---
>         call gridnl_hybrid (nx,lev,ncld
>      *  , cp,radsq,ut(1,1,jj),vt(1,1,jj),rdiv(1,1,jj)
129,134d120
< !
< ! no need of advect q
< !
< !     if( nislq ) then
< !       call nislq_advect(dt,ut,vt,pdot,pt,qt,qt)
< !     endif
154,163d139
< !hmhj: ok with 3,1 but it should be 2,0
<       if( nislq ) then
<       call joinrs(cc,diveng,ddtemp,dummy,dummy,nx,my_max,lev
<      &           ,jlistnum,3,1)
<       call tranrs(jtrun,jtmax,nx,my,my_max,lev,poly,weight,cc
<      &           ,wss,3,nsize)
<       call ujoinrs(wss,divten,temten,dummy,dummy,jtrun,jtmax,lev
```

```
<      &           ,mlistnum,3,1)
<       qten = 0.0
<       else
170d145
<       endif
174,177c149,151
<      &            ,weight,cim,onocos,poly,dpoly,temten,nsize)
<       if( .not. nislq )
<      &call rstrantq (jtrun,jtmax,nx,my,my_max,lev*ncld,qadvv,qadvu
<      &            ,weight,cim,onocos,poly,dpoly,qten,nsize)
---
>      *            ,weight,cim,onocos,poly,dpoly,temten,nsize)
>       call rstrantq (jtrun,jtmax,nx,my,my_max,lev*ncld,qadvv,qadvu
>      *            ,weight,cim,onocos,poly,dpoly,qten,nsize)
179c153
<      &            ,weight,cim,onocos,poly,dpoly,hldten,vorten,nsize)
---
>      *            ,weight,cim,onocos,poly,dpoly,hldten,vorten,nsize)
```

==========

< **gather_spec.f > gather_spec.f-orig**

```
3c3
<      2   uzm,lev,ncld,jtrun,jtmax,my,nsize,nislq)
---
>      2   uzm,lev,ncld,jtrun,jtmax,my,nsize)
15d14 <       logical nislq
30,31d28
<
<       if( .not. nislq ) then
34,35d30
<       endif
<
44,45d38
<
<       if( .not. nislq ) then
48,49d40
<       endif
<
58,59d48
<
<       if( .not. nislq ) then
62,63d50
<       endif
<
```

==========

< outflds.f > outflds.f-orig

```
123a124
>       j=jlist1(jj)
```

==========

< getrdy.f > getrdy.f-orig

```
493c493
<     4       , ggdef,gmdef,hybrid,nislq)
---
>     4       , ggdef,gmdef,hybrid)
510,517d509
<       if( nislq ) then
<       call joinrs(cc,tt,dummy,dummy,dummy,nx,my_max,lev,jlistnum,1,0)
<       call tranrs(jtrun,jtmax,nx,my,my_max,lev,poly,weight,cc
<      &          ,wss,1,nsize)
<       call ujoinrs(wss,temnow,dummy,dummy,dummy,jtrun,jtmax,lev
<      &          ,mlistnum,1,0)
<       qnow = 0.0
<       else
523d514
<       endif
526,528d516
<
< !     call check_maxmin  (temnow,lev*2,jtrun,mlistnum,' temnow after tranrs ')
< !     call check_maxmin  (plnow,1,jtrun,mlistnum,' plnow after tranrs1 ')
694,700d681
<       if( nislq ) then
<       call joinsr(wss,vornow,divnow,temnow,dummy,jtrun,jtmax,lev
<      *          ,mlistnum,3,0)
<       call transr(jtrun,jtmax,nx,my,my_max,lev,poly,wss,cc
<      *          ,3,nsize)
<       call ujoinsr(cc,rvor,rdiv,tt,dummy,nx,my_max,lev,jlistnum,3,0)
<       else
706d686
<       endif
901d880
< c
912d890
<
```

==========

< ujoinrs.f > ujoinrs.f-orig

```
6,45c6,11
<       if( ncld.eq.0 ) then
<       if(num .eq. 1) call ujoin10rs(wss,s1,jtrun,jtmax,lev,mlistnum)
<       if(num .eq. 2) call ujoin20rs(wss,s1,s2,jtrun,jtmax,lev,mlistnum)
```

```
<       else
<       if(num .eq. 2) call ujoin2rs(wss,s1,s2,jtrun,jtmax,lev,mlistnum,ncld)
<       if(num .eq. 3) call ujoin3rs(wss,s1,s2,s3,jtrun,jtmax,lev,mlistnum,ncld)
<       if(num .eq. 4) call ujoin4rs(wss,s1,s2,s3,s4,jtrun,jtmax,lev,mlistnum,ncld)
<       endif
< c
<       return
<       end
< c
<       subroutine ujoin10rs(wss,s1,jtrun,jtmax,lev,mlistnum)
< c
<       dimension wss (lev,2,1,jtrun,jtmax)
< c
<       dimension s1(lev,2,jtrun,jtmax)
< c
<       do 10 m=1,mlistnum
<       do 10 l=1,jtrun
<       do 10 k = 1, lev*2
<         s1(k,1,l,m) = wss(k,1,1,l,m)
<    10  continue
< c
<       return
<       end
< c
<       subroutine ujoin20rs(wss,s1,s2,jtrun,jtmax,lev,mlistnum)
< c
<       dimension wss (lev,2,2,jtrun,jtmax)
< c
<       dimension s1(lev,2,jtrun,jtmax)
<       dimension s2(lev,2,jtrun,jtmax)
< c
<       do 10 m=1,mlistnum
<       do 10 l=1,jtrun
<       do 10 k = 1, lev*2
<         s1(k,1,l,m) = wss(k,1,1,l,m)
<         s2(k,1,l,m) = wss(k,1,2,l,m)
<    10  continue
---
>       if(num .eq. 2)
>      &  call ujoin2rs(wss,s1,s2,jtrun,jtmax,lev,mlistnum,ncld)
>       if(num .eq. 3)
>      &  call ujoin3rs(wss,s1,s2,s3,jtrun,jtmax,lev,mlistnum,ncld)
>       if(num .eq. 4)
>      &  call ujoin4rs(wss,s1,s2,s3,s4,jtrun,jtmax,lev,mlistnum,ncld)

==========================================================
```

< **ujoinsr.f > ujoinsr.f-orig**

```
5,9d4
<       if( ncld.eq.0 ) then
<       if(num .eq. 1) call ujoin10sr(cc,r1,nx,my_max,lev,jlistnum)
<       if(num .eq. 2) call ujoin20sr(cc,r1,r2,nx,my_max,lev,jlistnum)
<       if(num .eq. 3) call ujoin30sr(cc,r1,r2,r3,nx,my_max,lev,jlistnum)
<       else
16,64d10
<       endif
< c
<       return
<       end
<
<       subroutine ujoin10sr(cc,r1,nx,my_max,lev,jlistnum)
< c
<       dimension cc(nx+3,lev,1,my_max)
<       dimension r1(nx,lev,my_max)
< c
<       do 10 jj =1, jlistnum
<       do 10 k=1,lev
<       do 10 i=1,nx
<         r1(i,k,jj)=cc(i,k,1,jj)
<    10 continue
< c
<       return
<       end
< c
<       subroutine ujoin20sr(cc,r1,r2,nx,my_max,lev,jlistnum)
< c
<       dimension cc(nx+3,lev,2,my_max)
<       dimension r1(nx,lev,my_max)
<       dimension r2(nx,lev,my_max)
< c
<       do 10 jj =1, jlistnum
<       do 10 k=1,lev
<       do 10 i=1,nx
<         r1(i,k,jj)=cc(i,k,1,jj)
<         r2(i,k,jj)=cc(i,k,2,jj)
<    10 continue
< c
<       return
<       end
< c
<       subroutine ujoin30sr(cc,r1,r2,r3,nx,my_max,lev,jlistnum)
< c
<       dimension cc(nx+3,lev,3,my_max)
```

```
<       dimension r1(nx,lev,my_max)
<       dimension r2(nx,lev,my_max)
<       dimension r3(nx,lev,my_max)
< c
<       do 10 jj =1, jlistnum
<       do 10 k=1,lev
<       do 10 i=1,nx
<         r1(i,k,jj)=cc(i,k,1,jj)
<         r2(i,k,jj)=cc(i,k,2,jj)
<         r3(i,k,jj)=cc(i,k,3,jj)
<    10 continue

===================================================================
< gridnl_hybrid.f    > gridnl_hybrid.f-orig
1,2c1,2
<       subroutine gridnl_hybrid ( nx,lev,ncld,nislq
<      &                 , cp,radsq,ut,vt,pdot,rdiv,rvor,tt,qt,phi
---
>       subroutine gridnl_hybrid ( nx,lev,ncld
>      &                 , cp,radsq,ut,vt,rdiv,rvor,tt,qt,phi
19d18
< c  pdot: gridpt vertical velocity in term of pressure sdot(dp/ds) ~ dp/dt
52,53c51
<       dimension ut(nx,lev),vt(nx,lev),pdot(nx,lev+1)
<      *, rdiv(nx,lev),rvor(nx,lev)
---
>       dimension ut(nx,lev),vt(nx,lev),rdiv(nx,lev),rvor(nx,lev)
70,71d67
< c
<       logical nislq
185,188d180
< c moisture advection
< c
<       if( .not. nislq ) then
< c
227,239d218
< c
< c end of moisture advection
< c
<       else
<
<       qvadv = 0.0
<       qadvv = 0.0
<       qadvu = 0.0
<       pdot(:,    1) = 0.0
<       pdot(:,lev+1) = 0.0
```

```
<       pdot(:,2:lev) = sd(:,2:lev)
<
<       endif

==========================================================================
< hdiffu.f    > hdiffu.f-orig
3c3
<      2                , eps4,trefs,qrefs,nislq )
---
>      2                , eps4,trefs,qrefs )
16d15
<       logical nislq
96d94
<       if( .not. nislq ) then
109d106
<       endif
122c119
<        call filter_top (jtrun,jtmax,lev,ncld,nislq,temnow,qnow,vornow,divnow)
---
>        call filter_top (jtrun,jtmax,lev,ncld,temnow,qnow,vornow,divnow)
129c126
<       subroutine filter_top(jtrun,jtmax,lev,ncld,nislq,temnow,qnow
---
>       subroutine filter_top(jtrun,jtmax,lev,ncld,temnow,qnow
144d140
<       logical nislq
174c170
< chmhj    qnow(k,1,n,m)  = qnow(k,1,n,m)*flt
---
>          qnow(k,1,n,m)  = qnow(k,1,n,m)*flt
178c174
< chmhj    qnow(k,2,n,m)  = qnow(k,2,n,m)*flt
---
>          qnow(k,2,n,m)  = qnow(k,2,n,m)*flt
184,200d179
<       if( .not. nislq ) then
<       do k = 1, ktop
<       do m = 1, mlistnum
<        mf=max(2,mlist(m))
<       do n = mf, jtrun
<        nflt = int ( wvn_top(k) / float(n) )
<        flt = min( 1.0, float(nflt) )
<         do l=1,ncld
<          kk=k+(l-1)*lev
<          qnow(kk,1,n,m)  = qnow(kk,1,n,m)*flt
<          qnow(kk,2,n,m)  = qnow(kk,2,n,m)*flt
```

```
<         end do
<       end do
<       end do
<       end do
<       end if
< c
213c192
< chmhj    qnow(k,1,n,m)  = qnow(k,1,n,m)*flt
---
>         qnow(k,1,n,m)  = qnow(k,1,n,m)*flt
217c196
< chmhj    qnow(k,2,n,m)  = qnow(k,2,n,m)*flt
---
>         qnow(k,2,n,m)  = qnow(k,2,n,m)*flt
222,238d200
<
<       if( .not. nislq ) then
<       do k = 1, ktop
<       do m = 1, mlistnum
<        mf=max(2,mlist(m))
<       do n = mf, jtrun
<          fac = min(wvn_top(k),float(n-1)) * pi / wvn_top(k)
<          flt = sin(fac)/fac
<          do l=1,ncld
<          kk=k+(l-1)*lev
<          qnow(kk,1,n,m)  = qnow(kk,1,n,m)*flt
<          qnow(kk,2,n,m)  = qnow(kk,2,n,m)*flt
<          end do
<       end do
<       end do
<       end do
<       end if

==========================================================
< incrini.f    > incrini.f-orig
106d105
<       if( nislq ) qtmp(:,1:lev,:) = max(0.0,qtmp(:,1:lev,:))
127d125
<       if( nislq ) qtmp(:,1:lev,:) = max(0.0,qtmp(:,1:lev,:))
156,163d153
<       if( nislq ) then
<       call joinrs(cc,tt,dumy,dummy,dummy,nx,my_max,lev,jlistnum,1,0)
<       call tranrs(jtrun,jtmax,nx,my,my_max,lev,poly,weight,cc
<      &          ,wss,1,nsize)
<       call ujoinrs(wss,temnow,dummy,dummy,dummy,jtrun,jtmax,lev
<      &          ,mlistnum,1,0)
```

```
<       qnow = 0.0
<       else
169d158
<       endif
204,210d192
<       if( nislq ) then
<       call joinsr(wss,vornow,divnow,temnow,dummy,jtrun,jtmax,lev
<      &          ,mlistnum,3,0)
<       call transr(jtrun,jtmax,nx,my,my_max,lev,poly,wss,cc,3,nsize)
<       call ujoinsr(cc,rvor,rdiv,tt,dummy,nx,my_max,lev,jlistnum,3,0)
<       qnow = 0.0
<       else
215d196
<       endif
```

==========

< **initial.f > initial.f-orig**

```
242c242
<      &           ,mlistnum,3,0)
---
>      &           ,mlistnum,3,1)
244c244
<         call ujoinsr(cc,rvor,rdiv,tt,dummy,nx,my_max,lev,jlistnum,3,0)
---
>         call ujoinsr(cc,rvor,rdiv,tt,dummy,nx,my_max,lev,jlistnum,3,1)
279d278
<       if( .not. nislq ) then
287d285
<       endif
```

==========

< **intgrt.f > intgrt.f-orig**

```
75a76
>
276d276
<         if( .not. nislq ) then
280,282d279
<         else
<          qbar = 0.0
<         endif
304d300
<       if( .not. nislq ) then
311,313d306
<       else
<       qten = 0.0
<       endif
```

```
339,340c332,333
<         call gridnl_hybrid (nx,lev,ncld,nislq
<      *  , cp,radsq,ut(1,1,jj),vt(1,1,jj),pdot(1,1,jj),rdiv(1,1,jj)
---
>         call gridnl_hybrid (nx,lev,ncld
>      *  , cp,radsq,ut(1,1,jj),vt(1,1,jj),rdiv(1,1,jj)
365,388d357
< !
< ! total advect q from qpm to qt
< !
<       if( nislq ) then
<         if( forward ) then
<           do jj = 1, jlistnum
<             do k = 1, lev*ncld
<               do i = 1, nx
<                qp (i,k,jj) = qt(i,k,jj)
<                qpm(i,k,jj) = qt(i,k,jj)
<               enddo
<             enddo
<           enddo
<         else
<           do jj = 1, jlistnum
<             do k = 1, lev*ncld
<               do i = 1, nx
<                qp(i,k,jj) = qt(i,k,jj)
<               enddo
<             enddo
<           enddo
<         endif
<         call nislq_advect(dta,ut,vt,pdot,pt,qpm,qt)
<       endif
405,414d373
< ! hmhj: ok with 3,1 in fact it should be 2,0
<       if( nislq ) then
<       call joinrs(cc,diveng,ddtemp,dummy,dummy,nx,my_max,lev
<      &          ,jlistnum,3,1)
<       call tranrs(jtrun,jtmax,nx,my,my_max,lev,poly,weight,cc
<      &          ,wss,3,nsize)
<       call ujoinrs(wss,divten,temten,dummy,dummy,jtrun,jtmax,lev
<      &          ,mlistnum,3,1)
<       qten = 0.0
<       else
421d379
<       endif
427,428c385
<       if( .not. nislq )
```

```
<      &call rstrantq (jtrun,jtmax,nx,my,my_max,lev*ncld,qadvv,qadvu
---
>      call rstrantq (jtrun,jtmax,nx,my,my_max,lev*ncld,qadvv,qadvu
432,433d388
< !
<      if( nislq )  qten = 0.0
478,482d432
<      if( nislq ) then
<      call zimadv_nislq ( my,my_max,lev
<     &            ,jtrun,jtmax,dt1,poly,onocos,weight
<     &            ,uzm,vorten,vornow,vorold,nsize  )
<      else
486d435
<      endif
550,551d498
<
<      if( .not. nislq ) then
557d503
<      endif
581d526
<      if( .not. nislq ) then
588,590d532
<      else
<       qnow = 0.0
<      endif
605c547
<     2                    , qnow,eps4,trefs,qrefs,nislq )
---
>     2                    , qnow,eps4,trefs,qrefs )
616,622d557
<      if( nislq ) then
<      call joinsr(wss,temnow,dummy,dummy,dummy,jtrun,jtmax,lev
<     *           ,mlistnum,1,0)
<      call transr(jtrun,jtmax,nx,my,my_max,lev,poly,wss,cc,1,nsize)
<      call ujoinsr(cc,tt,dummy,dummy,dummy,nx,my_max,lev,jlistnum,1,0)
<      qnow = 0.0
<      else
627d561
<      endif
660,667d593
<      if( nislq ) then
<      call joinrs(cc,tt,dummy,dummy,dummy,nx,my_max,lev,jlistnum,1,0)
<      call tranrs(jtrun,jtmax,nx,my,my_max,lev,poly,weight,cc
<     *           ,wss,1,nsize)
<      call ujoinrs(wss,temnow,dummy,dummy,dummy,jtrun,jtmax,lev
<     *           ,mlistnum,1,0)
```

```
<       qnow = 0.0
<       else
673d598
<       endif
727d651
<       if( .not. nislq ) then
742,744d665
<       else
<           qten = 0.0
<       endif
757d677
<       if( .not. nislq ) then
764,766d683
<       else
<        qten = 0.0
<       endif
788c705
<      2                    , qten,eps4,trefs,qrefs,nislq )
---
>      2                    , qten,eps4,trefs,qrefs )
799,805d715
<       if( nislq ) then
<       call joinsr(wss,temten,dummy,dummy,dummy,jtrun,jtmax,lev
<      *           ,mlistnum,1,0)
<       call transr(jtrun,jtmax,nx,my,my_max,lev,poly,wss,cc,1,nsize)
<       call ujoinsr(cc,tt,dummy,dummy,dummy,nx,my_max,lev,jlistnum,1,0)
<       qten = 0.0
<       else
810d719
<       endif
841,848d749
<       if( nislq ) then
<       call joinrs(cc,tt,dummy,dummy,dummy,nx,my_max,lev,jlistnum,1,0)
<       call tranrs(jtrun,jtmax,nx,my,my_max,lev,poly,weight,cc
<      *           ,wss,1,nsize)
<       call ujoinrs(wss,temten,dummy,dummy,dummy,jtrun,jtmax,lev
<      *           ,mlistnum,1,0)
<       qten = 0.0
<       else
854d754
<       endif
897d796
<       if( .not. nislq ) then
906,917d804
<       else
<         qold = 0.0
```

```
<         qnow = 0.0
<         do jj = 1, jlistnum
<           do k = 1, lev*ncld
<             do i = 1, nx
<              qpm(i,k,jj) = qp(i,k,jj) + tfilt*( qpm(i,k,jj)
<     &              - 2.0*qp(i,k,jj) + qt(i,k,jj) )
<             enddo
<           enddo
<         enddo
<       endif
941,947d827
<       if( nislq ) then
<       call joinsr(wss,vornow,divnow,temnow,dummy,jtrun,jtmax,lev
<      *          ,mlistnum,3,0)
<       call transr(jtrun,jtmax,nx,my,my_max,lev,poly,wss,cc,3,nsize)
<       call ujoinsr(cc,rvor,rdiv,tt,dummy,nx,my_max,lev,jlistnum,3,0)
<       qnow = 0.0
<       else
952d831
<       endif
1090c969
<      2   uzm,lev,ncld,jtrun,jtmax,my,nsize,nislq)
---
>      2   uzm,lev,ncld,jtrun,jtmax,my,nsize)

================================================================
< joinrs.f    > joinrs.f-orig
5,8d4
<       if( ncld.eq.0 ) then
<       if(num .eq. 1) call join10rs(cc,r1,nx,my_max,lev,jlistnum)
<       if(num .eq. 2) call join20rs(cc,r1,r2,nx,my_max,lev,jlistnum)
<       else
11,12d6
<       if(num .eq. 4) call join4rs(cc,r1,r2,r3,r4,nx,my_max,lev,jlistnum,ncld)
<       endif
17,46d10
<       subroutine join10rs(cc,r1,nx,my_max,lev,jlistnum)
< c
<       dimension cc(nx+3,lev,1,my_max)
<       dimension r1(nx,lev,my_max)
< c
<       do 10 jj =1, jlistnum
<       do 10 k=1,lev
<       do 10 i=1,nx
<       cc(i,k,1,jj)= r1(i,k,jj)
<    10 continue
```

```
< c
<       return
<       end
< c
<       subroutine join20rs(cc,r1,r2,nx,my_max,lev,jlistnum)
< c
<       dimension cc(nx+3,lev,2,my_max)
<       dimension r1(nx,lev,my_max)
<       dimension r2(nx,lev,my_max)
< c
<       do 10 jj =1, jlistnum
<       do 10 k=1,lev
<       do 10 i=1,nx
<       cc(i,k,1,jj)= r1(i,k,jj)
<       cc(i,k,2,jj)= r2(i,k,jj)
<    10 continue
< c
<       return
<       end
< c
60c24
<       do 20 n=1,max(1,ncld)
---
>       do 20 n=1,ncld
86c50
<       do 20 n=1,max(1,ncld)
---
>       do 20 n=1,ncld
92,119d55
<    20 continue
< c
<       return
<       end
< c
<       subroutine join4rs(cc,r1,r2,r3,r4,nx,my_max,lev,jlistnum,ncld)
< c
<       dimension cc(nx+3,lev,2+ncld,my_max)
<       dimension r1(nx+3,lev,my_max)
<       dimension r2(nx+3,lev,my_max)
<       dimension r3(nx+3,lev,my_max)
<       dimension r4(nx+3,lev*ncld,my_max)
< c
<       do 10 jj =1, jlistnum
<       do 10 k=1,lev
<       do 10 i=1,nx
<       cc(i,k,1,jj)= r1(i,k,jj)
```

```
<       cc(i,k,2,jj)= r2(i,k,jj)
<       cc(i,k,3,jj)= r3(i,k,jj)
<    10 continue
< c
<       do 20 jj =1, jlistnum
<       do 20 n=1,max(1,ncld)
<       nk=(n-1)*lev
<       do 20 k=1,lev
<       kk=nk+k
<       do 20 i=1,nx
<       cc(i,k,3+n,jj)= r4(i,kk,jj)

===================================================================
< joinsr.f  > joinsr.f_org
6,10d5
<       if( ncld.eq.0 ) then
<       if(num .eq. 1) call join10sr(wss,s1,jtrun,jtmax,lev,mlistnum)
<       if(num .eq. 2) call join20sr(wss,s1,s2,jtrun,jtmax,lev,mlistnum)
<       if(num .eq. 3) call join30sr(wss,s1,s2,s3,jtrun,jtmax,lev,mlistnum)
<       else
17,71d11
<       endif
< c
<       return
<       end
< c
<       subroutine join10sr(wss,s1,jtrun,jtmax,lev,mlistnum)
< c
<       dimension wss (lev,2,1,jtrun,jtmax)
< c
<       dimension s1(lev,2,jtrun,jtmax)
< c
<       wss=0.
<       do 10 m=1,mlistnum
<       do 10 l=1,jtrun
<       do 10 k = 1, lev*2
<         wss(k,1,1,l,m) = s1(k,1,l,m)
<    10  continue
< c
<       return
<       end
< c
<       subroutine join20sr(wss,s1,s2,jtrun,jtmax,lev,mlistnum)
< c
<       dimension wss (lev,2,2,jtrun,jtmax)
< c
```

```
<       dimension s1(lev,2,jtrun,jtmax)
<       dimension s2(lev,2,jtrun,jtmax)
< c
<       wss=0.
<       do 10 m=1,mlistnum
<       do 10 l=1,jtrun
<       do 10 k = 1, lev*2
<         wss(k,1,1,l,m) = s1(k,1,l,m)
<         wss(k,1,2,l,m) = s2(k,1,l,m)
<    10  continue
< c
<       return
<       end
< c
<       subroutine join30sr(wss,s1,s2,s3,jtrun,jtmax,lev,mlistnum)
<
<       dimension wss (lev,2,3,jtrun,jtmax)
< c
<       dimension s1(lev,2,jtrun,jtmax)
<       dimension s2(lev,2,jtrun,jtmax)
<       dimension s3(lev,2,jtrun,jtmax)
< c
<       wss=0.
<       do 10 m=1,mlistnum
<       do 10 l=1,jtrun
<       do 10 k = 1, lev*2
<         wss(k,1,1,l,m) = s1(k,1,l,m)
<         wss(k,1,2,l,m) = s2(k,1,l,m)
<         wss(k,1,3,l,m) = s3(k,1,l,m)
<    10  continue
```

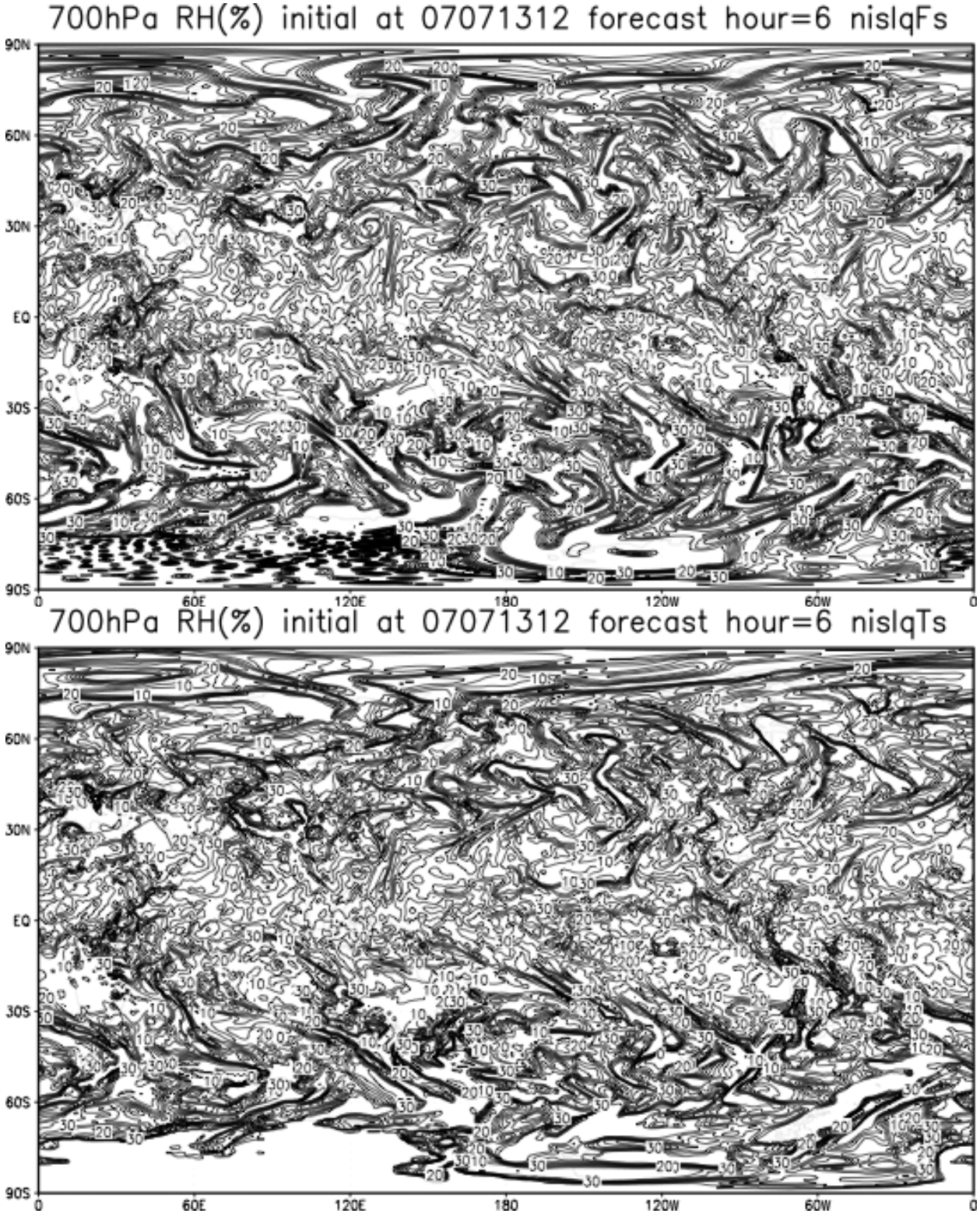

Fig. 1 Relative humidity on 700 hPa after 6 hr integration is obtained from Eulerian advection on top panel (nislqFs) and semi-Lagrangian tracer advection on bottom panel (nislqTs).

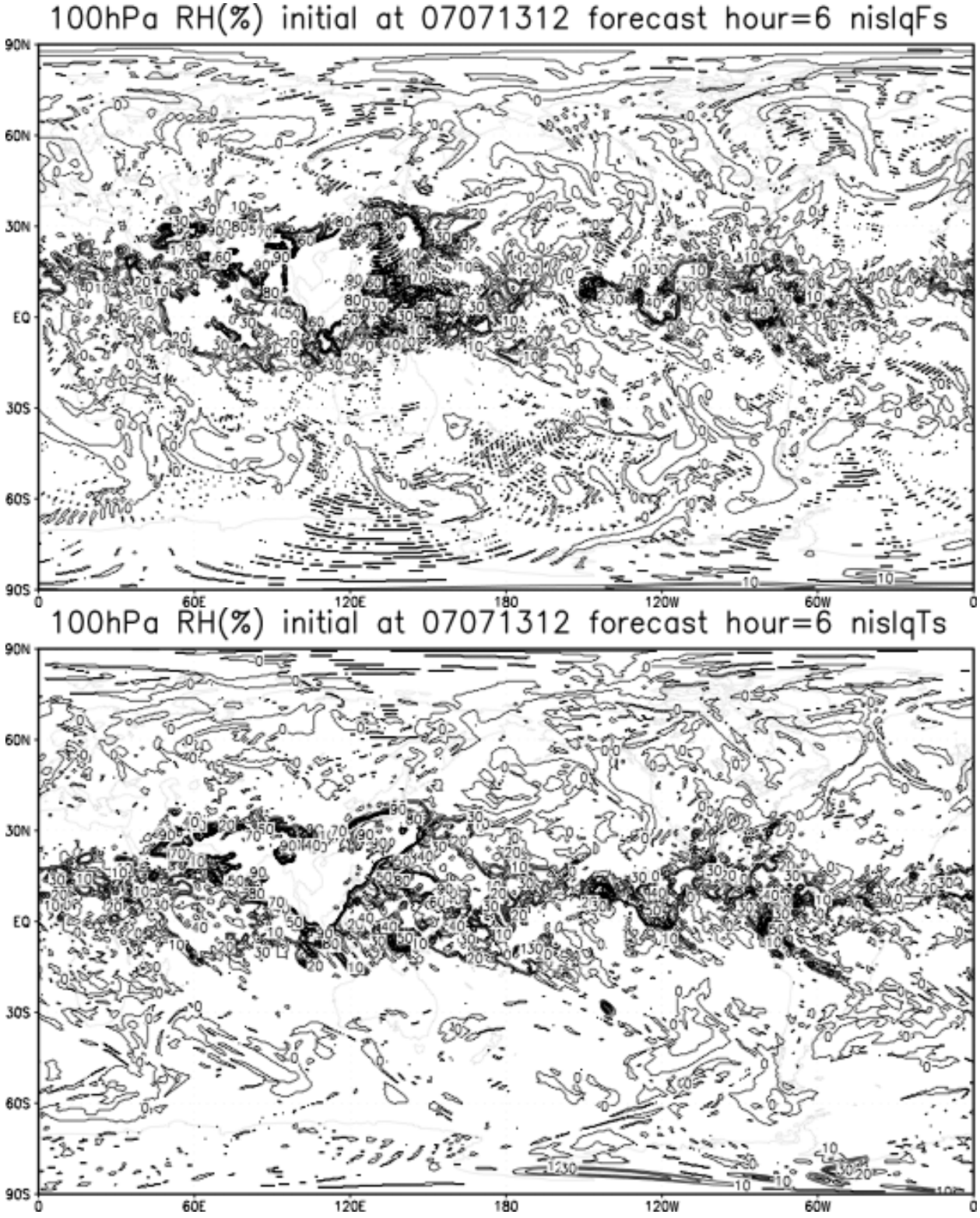

Fig. 2 The same as Fig. 1 except on 100 hPa.

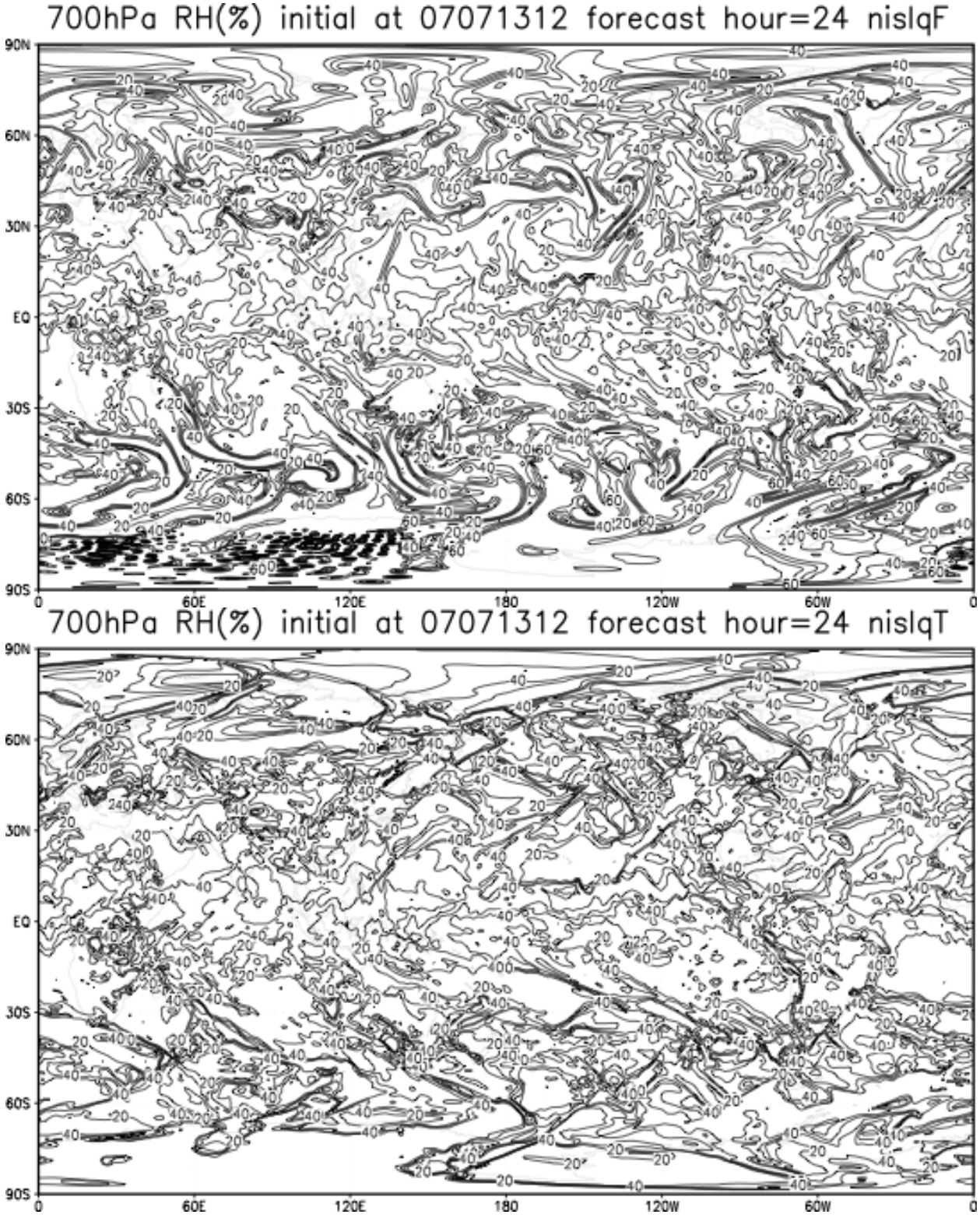

Fig. 3 The same as Fig. 1 except after 24 hr integration.

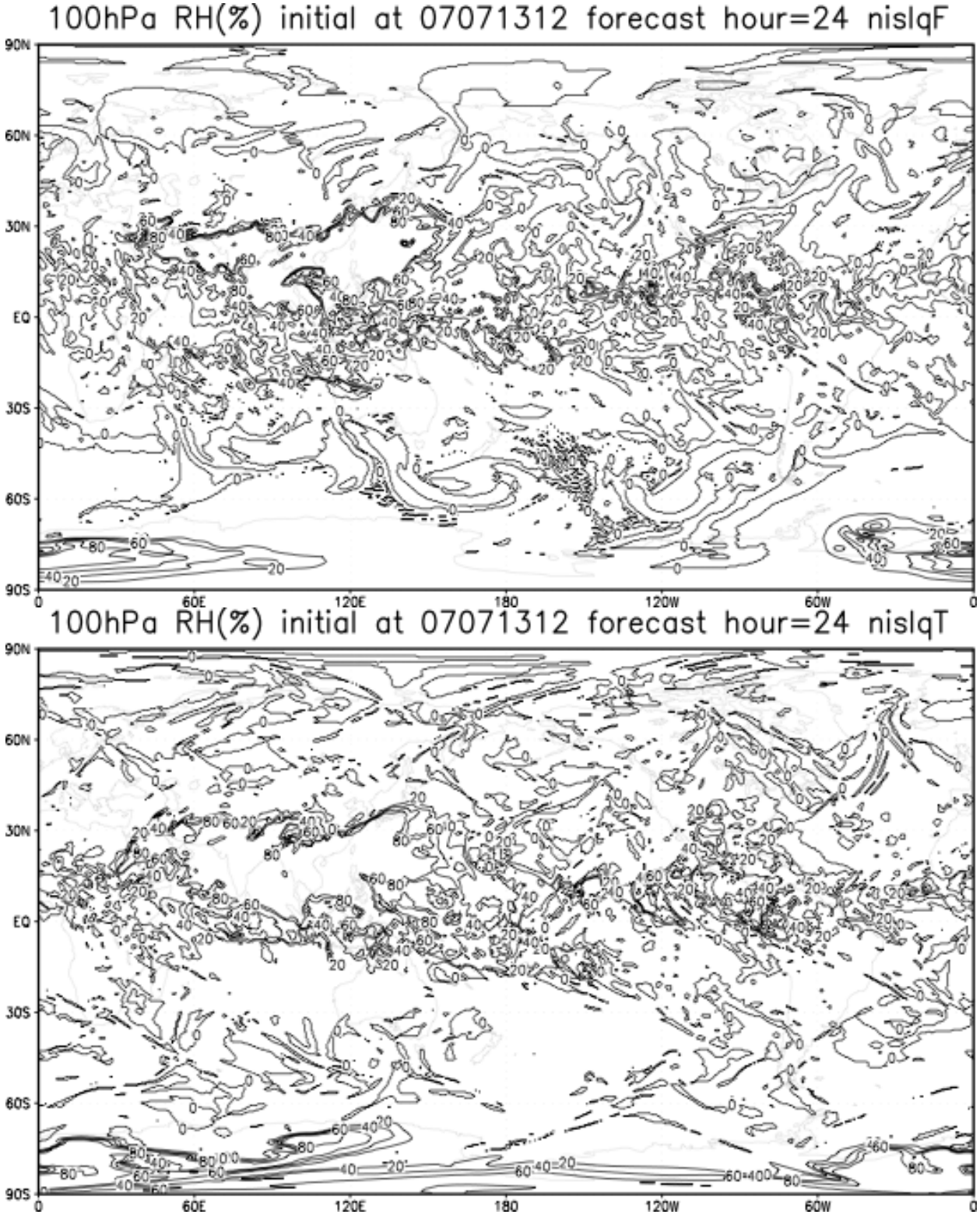

Fig. 4 The same as Fig. 2 except after 24 hr integration.

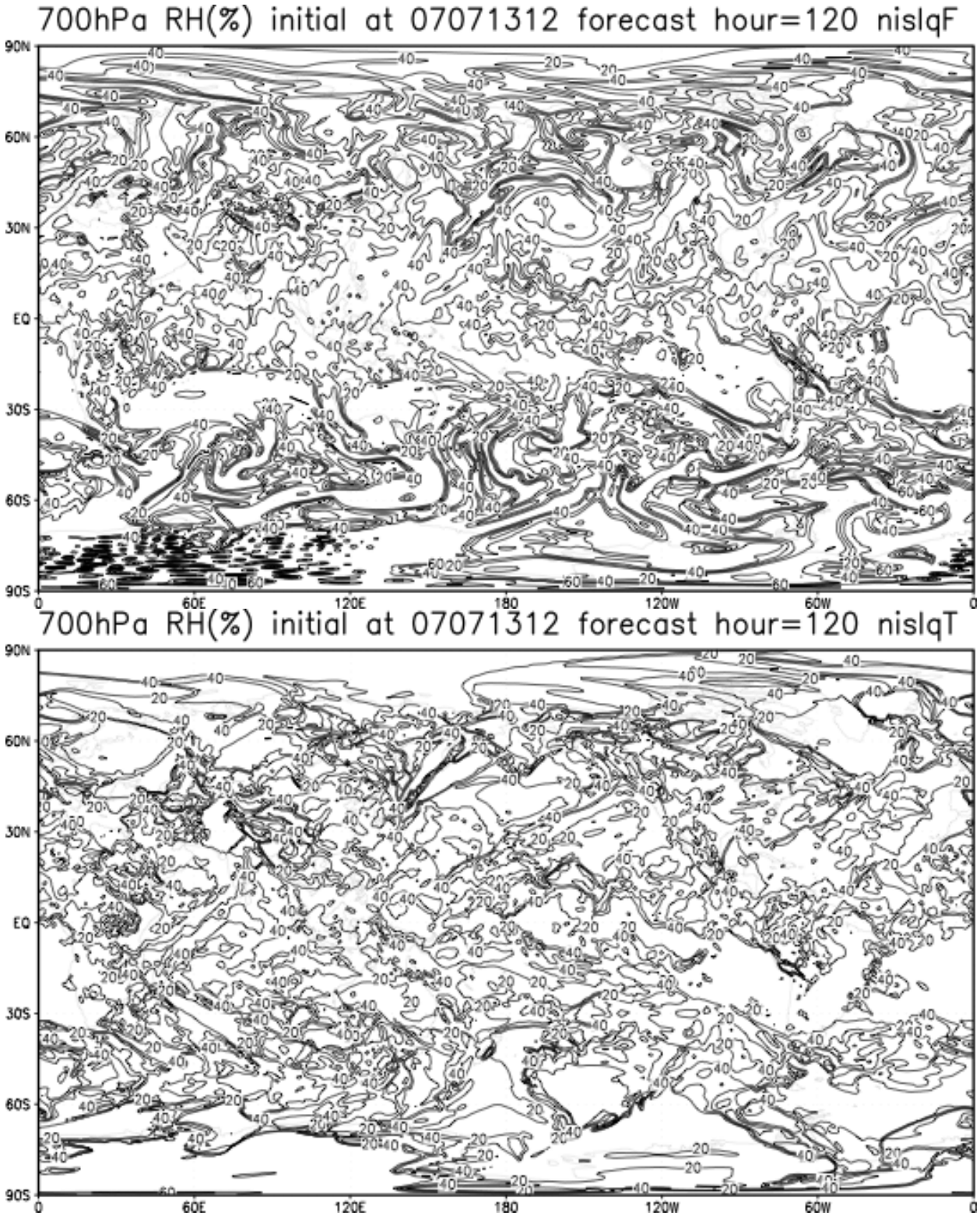

Fig. 5 The same as Fig. 1 except after 120 hr integration.

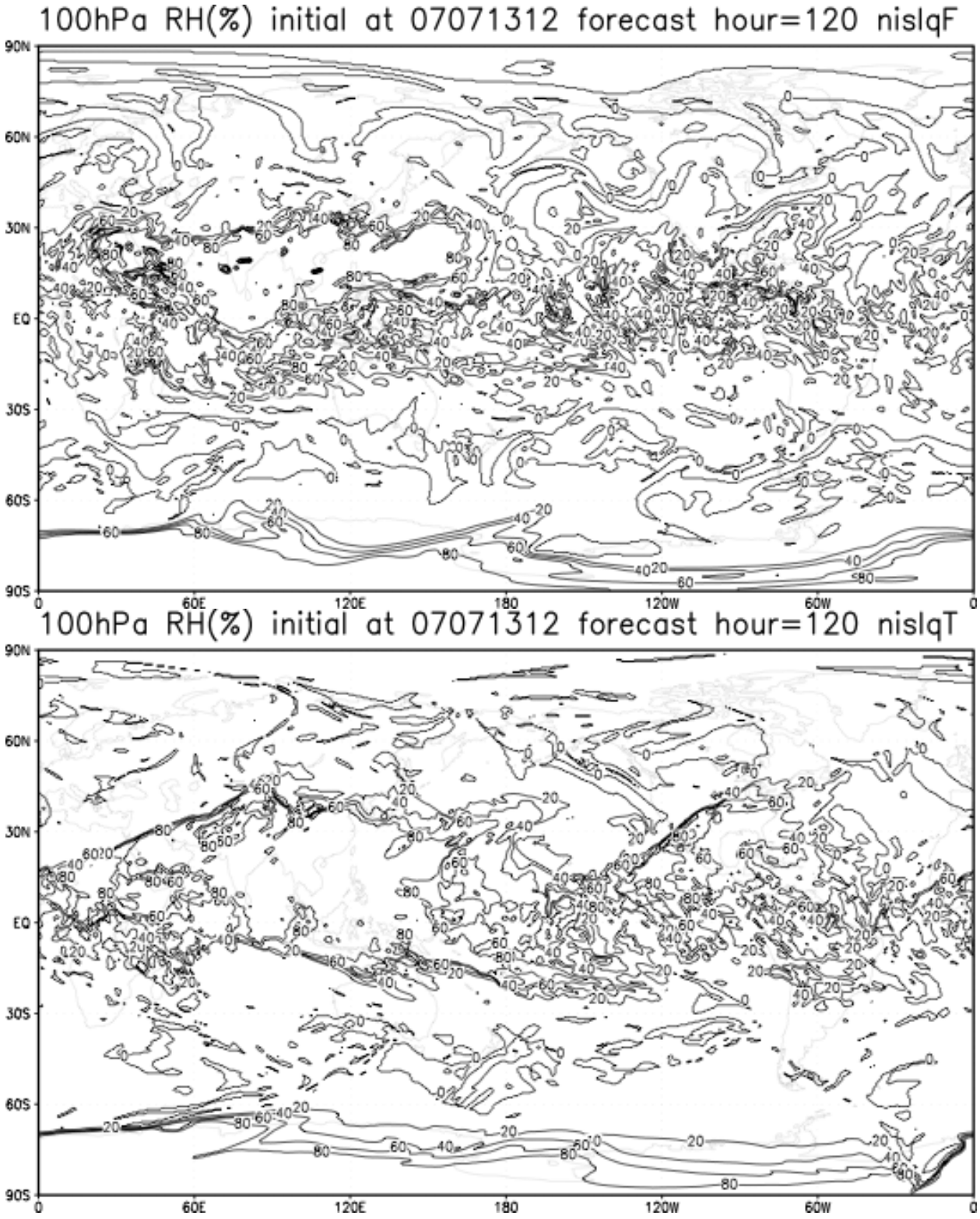

Fig. 6 The same as Fig. 2 except after 120 hr integration.

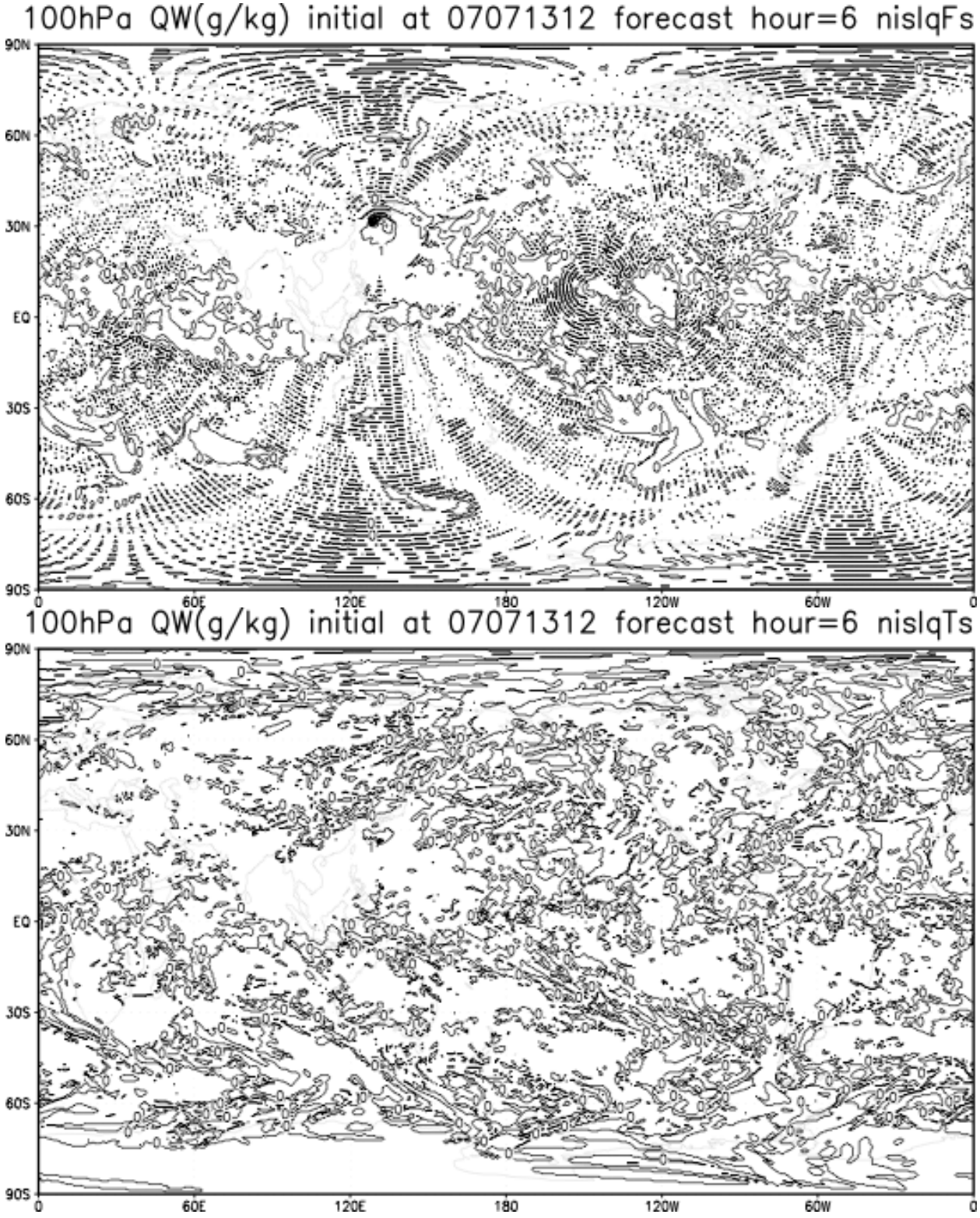

Fig. 7 Cloud water in g/kg on 100 hPa after 6 hr integration is obtained from Eulerian advection on top panel (nislqFs) and semi-Lagrangian advection on bottom panel (nislqTs).

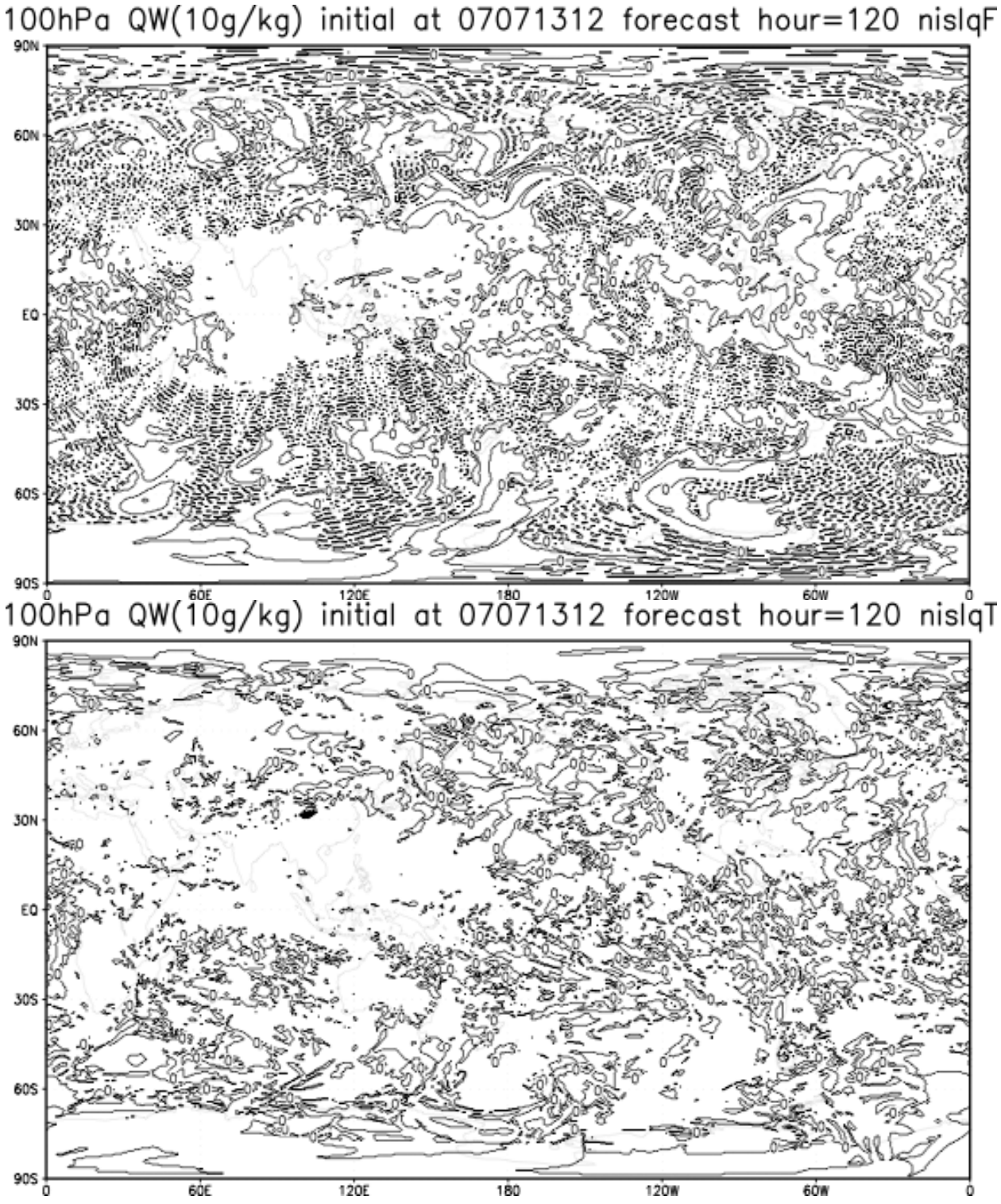

Fig. 8 The same as Fig. 7 except after 120 hr integration.

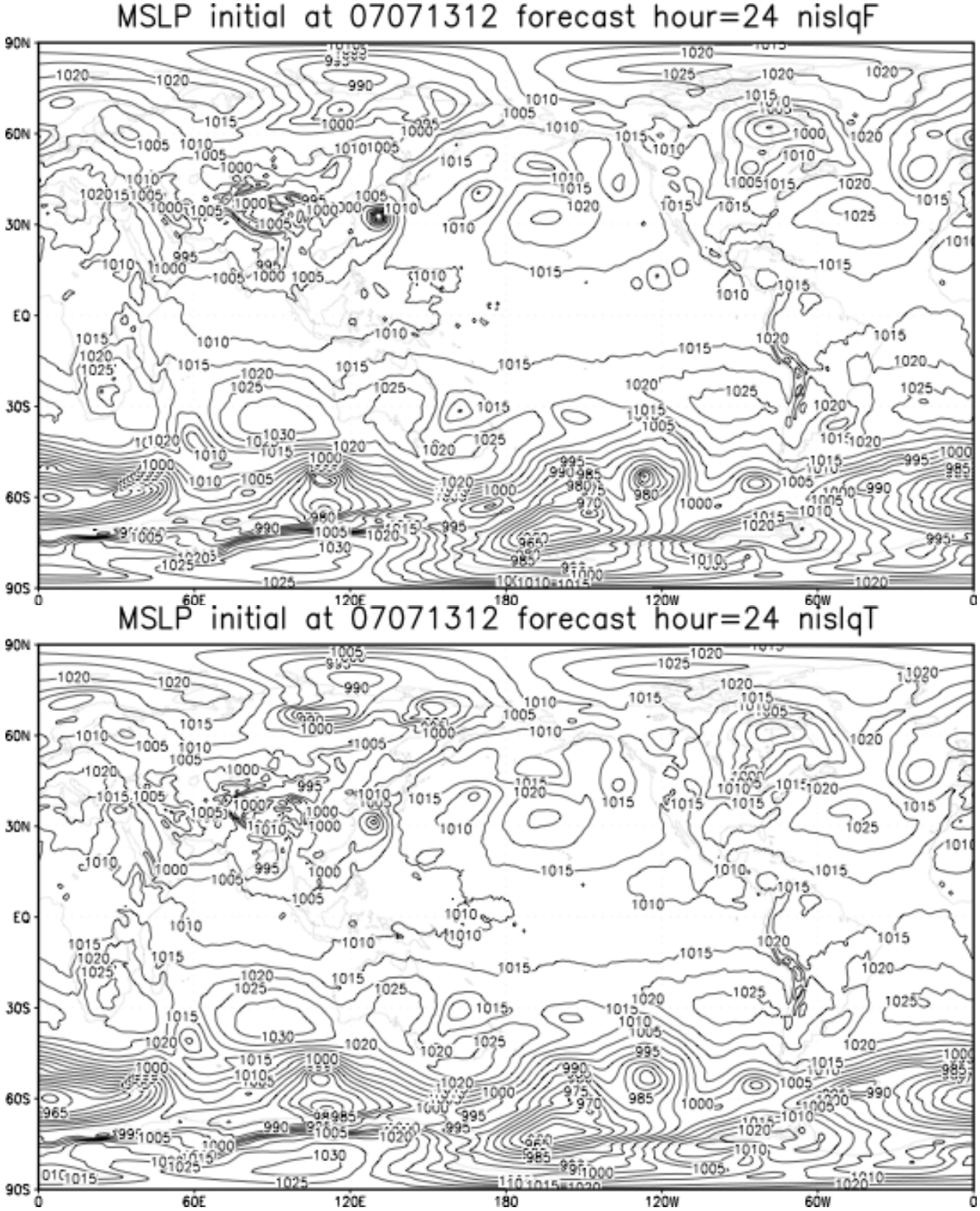

Fig. 9 Mean sea level pressure in hPa after 24 hr integration is obtained from Eulerian advection on top panel (nislqF) and semi-Lagrangian tracer advection on bottom panel (nislqT).

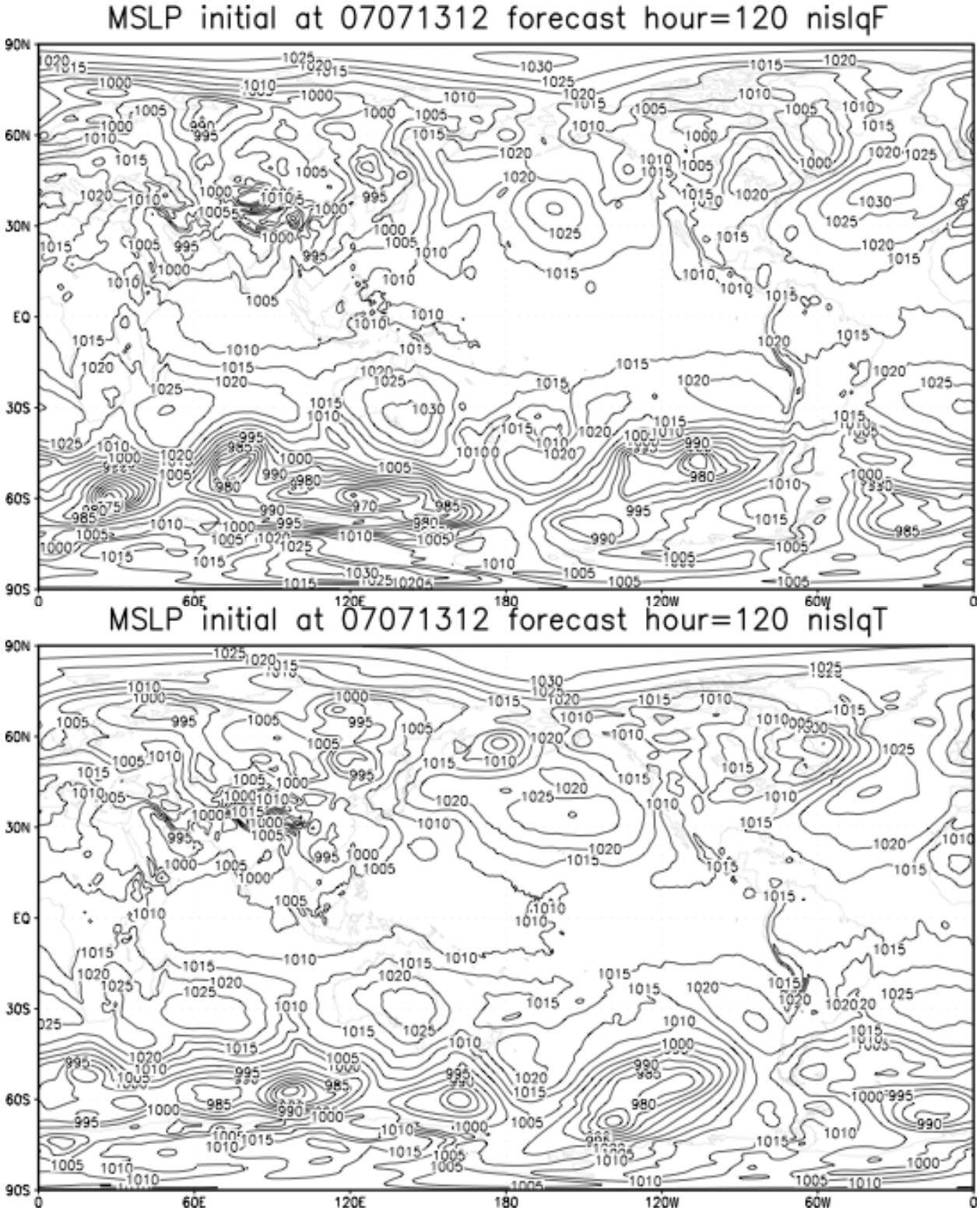

Fig. 10 The same as Fig. 9 except after 120 hr integration.

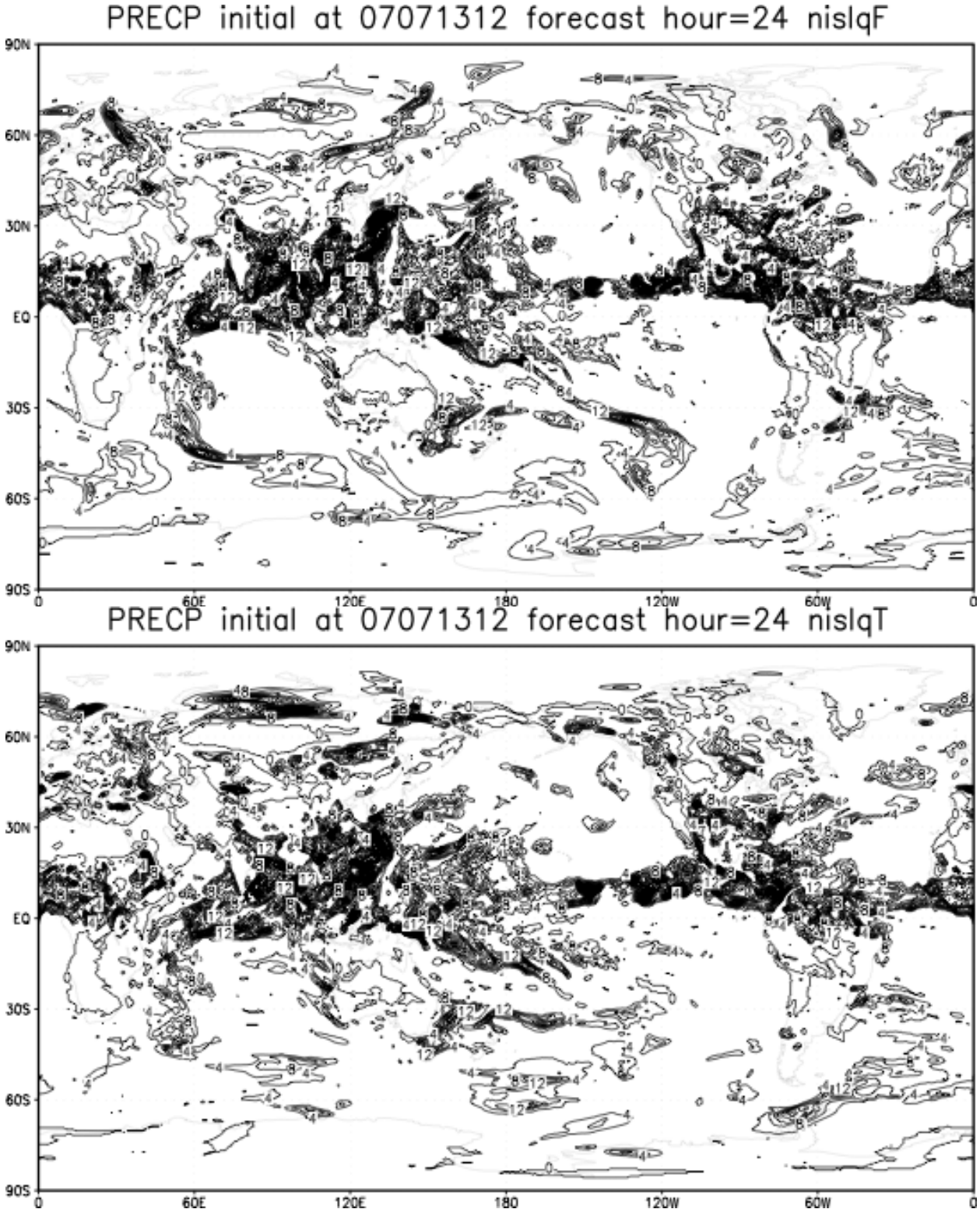

Fig. 11 The same as 9 except for accumulated rainfall in mm/day.

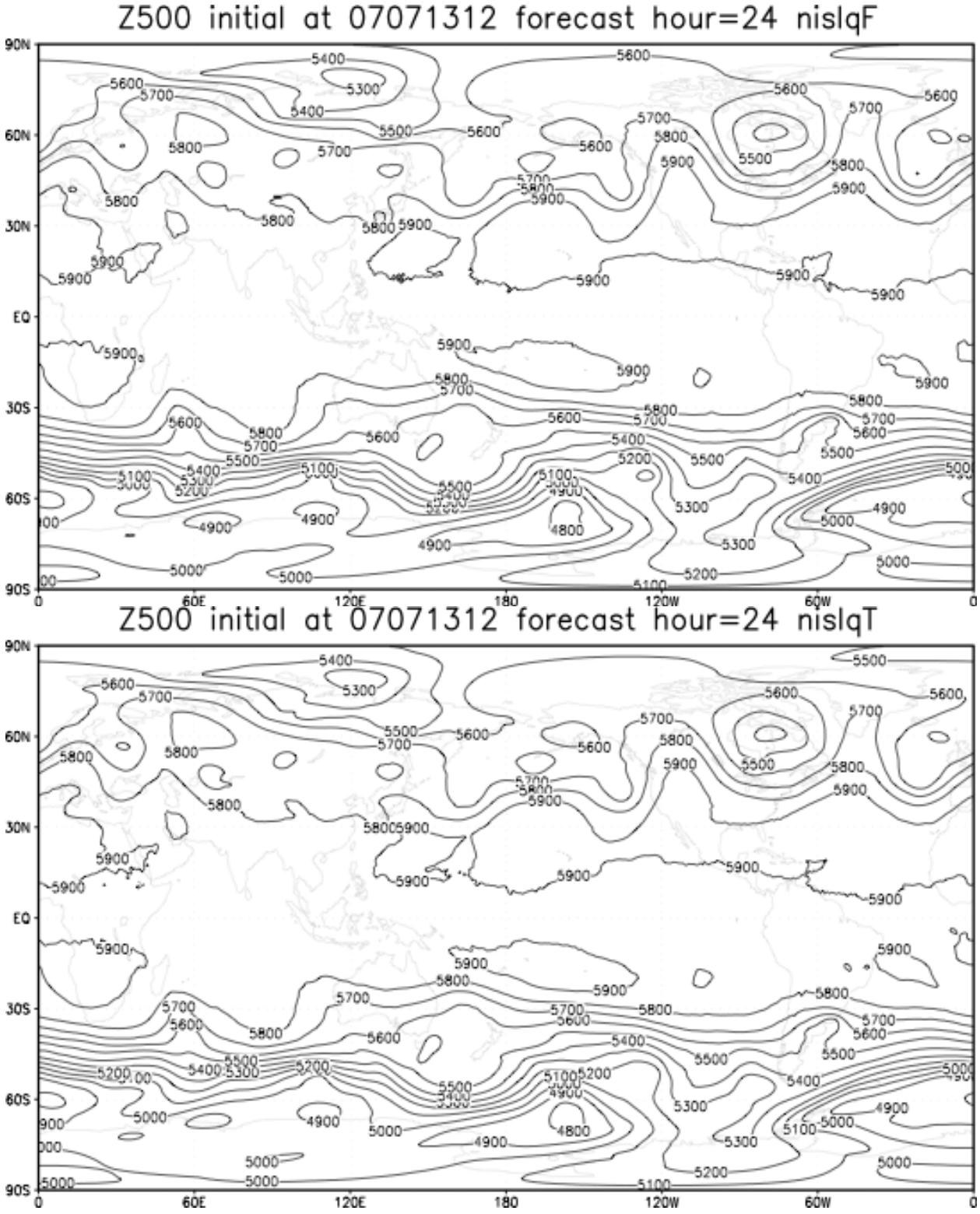

Fig. 12 Geopotential height in meter on 500 hPa after 24 hr integration is obtained from Eulerian advection on top panel (nislqF) and semi-Lagrangian tracer advection on bottom panel (nislqT).

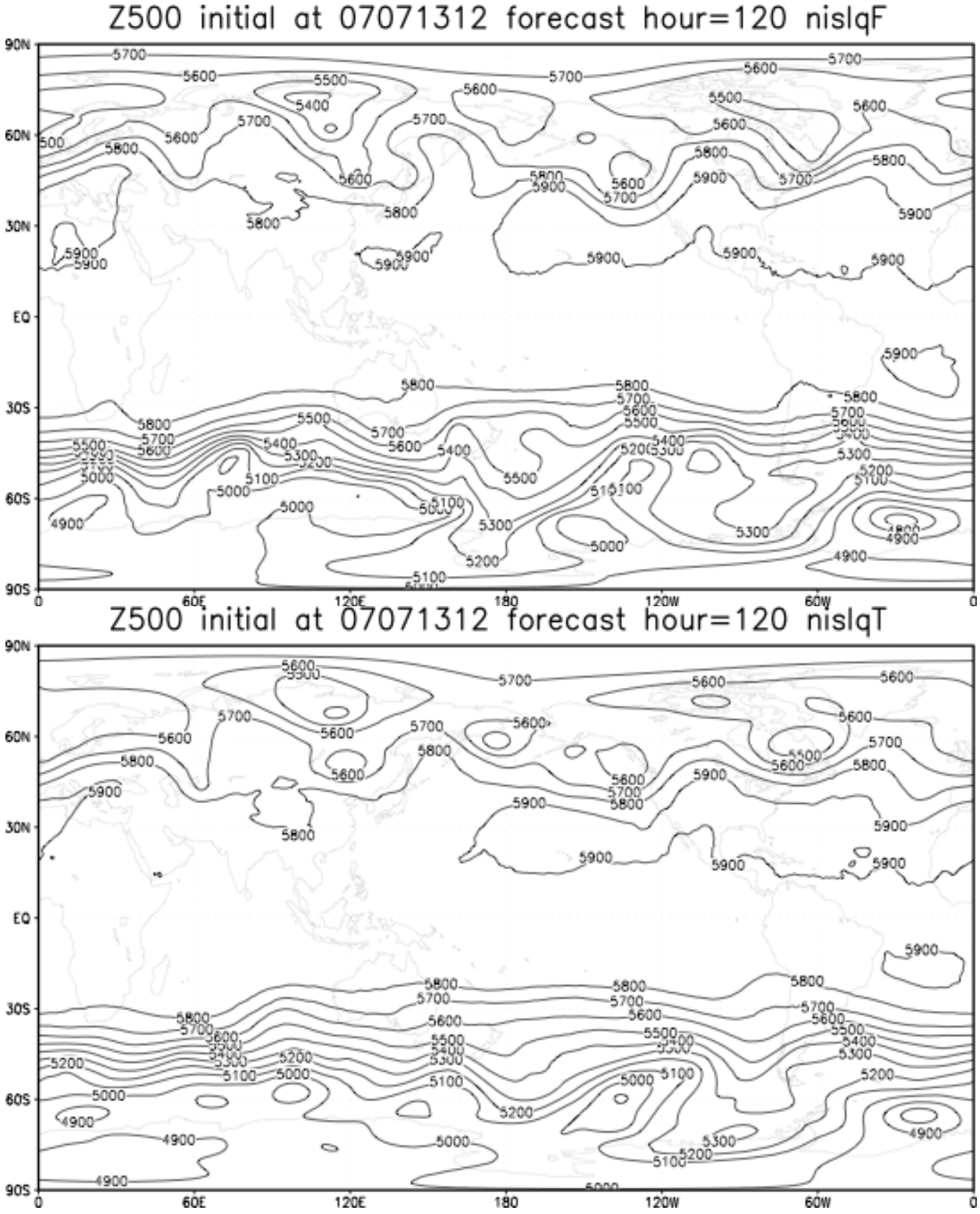

Fig. 13 The same as Fig. 12 except after 120 hr integration.