Proceedings of *XII CBMET-Brazilian Congress of Meteorology* , Foz de Iguaçú, PR, Brazil, August 2002

Optimized Neural Network Code for Data Assimilation

Nandamudi Lankalapalli. Vijaykumar[1], Stephan Stephany[1], Airam Jônatas Preto[1], Haroldo Fraga de Campos Velho[1] and Alexandre Nowosad[2]

[1]Laboratório Associado de Computação e Matemática Aplicada (LAC)
[2]Centro de Previsão de Tempo e Estudos Climáticos (CPTEC)
Instituto Nacional de Pesquisas Espaciais (INPE)
São José dos Campos, SP, Brasil
{vijay, stephan, airam, haroldo}@lac.inpe.br; alex@cptec.inpe.br

Abstract

The data assimilation process can be described as a procedure that uses observational data to improve the prediction made by an inaccurate mathematical model. Recently, neural networks have been proposed as a new method for data assimilation. The Multilayer Perceptron network with backpropagation learning was chosen for this procedure. Neural networks are inherently a parallel procedure. This paper presents some strategies being used to achieve an optimized parallel code for the network training. Code optimizations include the use of either High Performance Fortran directives or Message Passing Interface library calls. A neural network for Data Assimilation was trained based on both the physical models of the Lorenz and shallow water equations.

Introduction

Data Assimilation is a very important process in the numerical weather forecast. It permits the imbedding of observational data in the meteorological model. This data provides a feedback during the generation of the forecast in a real time fashion. However, the process of embedding the observational data is not straightforward and it has to be done in a very smooth manner in order not to disturb the forecast model thus leading to erroneous results. Classically, the assimilation process can be outlined as a two step process (Yang & Cotton, 1998):

Forecast step: $$w_n^f = F[w_{n-1}^a]$$

Analysis step: $$w_n^a = w_n^f + d_n$$

where w_n represents model state variable at time step n; $F[.]$ is the mathematical (forecast) model, superscripts f and a denote forecast and analyzed values respectively, and d_n is the innovation.

Several methods of data assimilation have been developed for air quality problems (Zannetti, 1990), numerical weather prediction (Daley, 1991), and numerical oceanic simulation (Bennet, 1992). In the case of atmospheric continuous data assimilation there are many deterministic and probabilistic methods (Daley, 1991). Deterministic methods include dynamic relaxation, variational methods and Laplace transform, whereas probabilistic methods include optimal interpolation and Kalman Filtering. Dynamic relaxation assumes the prediction model to be perfect, as does Laplace transform. Variational methods and optimal interpolation can be regarded as minimum-meansquare estimation of the atmosphere. In Kalman filtering the analysis innovation d_n is computed as a linear function of the misfit between observation (superscript *o*) and forecast (superscript *f*):

$$d_n = G_n (w_n^o - H_n w_n^f) \qquad (1)$$

where G_n is a weighting (gain) matrix, w_n^o is the observed value of w_n and H_n is an observation matrix. An adaptive extended Kalman filter has been tested in strongly nonlinear dynamical systems for assimilation procedure: the Lorenz chaotic system. Kalman filtering has the advantage of minimizing the error in the assimilation *plus* propagating the error itself from one data insertion to the next. But it is computationally too expensive as this process involves a heavy computational load, specially for large meteorological systems. A strategy to alleviate this load is the use of neural networks to emulate the performance of Kalman filtering with economy in computer time (Nowosad et al, 2000a). Neural networks (Haykin, 1994) can be efficiently applied to map two sets of data. Several architectures have been proposed for neural networks out of which Multilayer Perceptron with backpropagation learning (Haykin, 1994) may be mentioned.

In a recent paper, Gardner and Dorling (1998) did a survey on applications of the ANN in meteorology, where a brief introduction about ANN and the back-propagation algorithm are shown. It also cites applications in the atmospheric sciences looking at prediction (air-quality: surface ozone concentration, sulfur dioxide concentrations; severe weather; Indian monsoon, Brazilian rainfall anomalies, solar radiation), function approximation (air-quality, modeling of non-linear transfer functions), and pattern classification (cloud classification; distinction between clouds and ice or snow; classification of atmospheric circulation patterns; land cover classification; classification of convergence lines from radar imagery; etc). Although, the use of ANN for data assimilation can be understood as function approximation, this application was not mentioned in Gardner and Dorling's paper.

In this paper a Neural network with *backpropagation* learning was chosen for data assimilation. It basically consists of an input layer and an output layer with a number of hidden layers that may contain one or more neurons. As both the input as well as the expected output are fed with data to train the network, this process is known as supervised learning. The paper shows the strategies used to optimize the training phase code. The training was applied on two physical models: chaotic Lorenz's system (Lorenz, 1963) and shallow water model (Lynch, 1984). Optimizations included the use of HPF (High Performance Fortran) (HPF Forum, 1993) parallel directives or calls to the MPI (Message Passing Interface) library functions (Pacheco, 1996). The next section provides a brief introduction to Neural networks. The following section discusses the neural network architecture used for Data Assimilation application and presents in details the strategies used for achieving a parallel code during the training phase. The final section concludes with some comments and remarks.

Neural Networks

An artificial neural network (ANN) is an arrangement of units characterized by:

- a large number of very simple neuron-like processing units;
- a large number of weighted connections between the units, where the knowledge of a network is stored;
- highly parallel, distributed control.

The processing element (unit) in an ANN is a linear combiner with multiple weighted inputs, followed by an activation function. There are several different architectures of ANNs, most of which directly depend on the learning strategy adopted. It is not the aim of this paper to present an overview on ANN. Instead, a brief description of the ANN used is focused: the Multilayer Perceptron with backpropagation learning (Haykin, 1994).

The multilayer perceptron with backpropagation learning, also called the backpropagation neural network, is a feedforward network composed of an input layer, an output layer, and a number of hidden layers for extracting high order statistics from the input data (Haykin, 1994, page 19). Figure 1 shows a backpropagation neural network with one hidden layer. Functions g and F_{ANN} provide the activation for the hidden layer and the output layer, respectively. In order to make the network more flexible to solve nonlinear problems, the activation functions for the hidden layer are sigmoid functions.

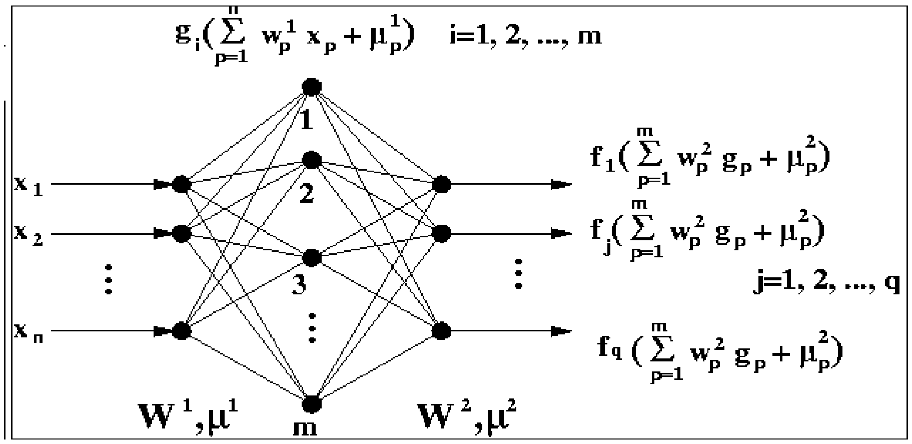
$g_i(\sum_{p=1}^{n} w_p^1 x_p + \mu_p^1)$ $i=1, 2, ..., m$
x_1
x_2
x_n
1
2
3
m
W^1, μ^1
W^2, μ^2
$f_1(\sum_{p=1}^{m} w_p^2 g_p + \mu_p^2)$
$f_j(\sum_{p=1}^{m} w_p^2 g_p + \mu_p^2)$
$j=1, 2, ..., q$
$f_q(\sum_{p=1}^{m} w_p^2 g_p + \mu_p^2)$

Figure 1. Multilayer Perceptron with one hidden layer with *m* neurons

Mathematically, a perceptron network simply maps input vectors of real values onto output vector of real values. The connections in the figure have associated weights that are adjusted during learning process, thus changing the performance of the network.

There are two distinct phases in the usage of an ANN: the training phase (learning process) and the running phase (activation of the network). In the training phase, the weights are adjusted for the best performance of the network in establishing the mapping of many input-output vector pairs. Once trained, the weights are fixed and new inputs can be presented to the network for it to compute corresponding outputs, based on what it has learned.

The training phase of a multilayer perceptron is controlled by a supervised learning algorithm, which differs from unsupervised learning. The main difference is that the latter uses only information contained in the input data, whereas the former requires both input and output (desired) data, which permits the calculation of the error of the network as the difference between the calculated output and the desired vector. Adjustment of the network's weights is conducted by backpropagating such error through the network. This adjustment is called *Backpropagation Algorithm*. The weight change rule is a development of the Perceptron learning rule. Weights are changed by an amount proportional to the error at that unit, times the output of the unit feeding into the weight. This is the essence of the so-called *delta rule*. The training phase can make use of two fashions: *batch mode* and *sequential mode* (Haykin, 1994). The former deals with the whole input data whereas the latter carries out the training based on each input pattern. The scope of this paper is restricted to batch mode in which *all* the input examples are taken at once and the learning procedure searches a set of weights θ and biases μ that minimizes the total squared error:

$$e_m = \sum_{k=1}^{N} \left\| F_{ANN}(X_k, \theta, \mu, m) - F(X_k) \right\|_2 \qquad (2)$$

where N is the number of examples in the training set, X_k is the input vector of example k, θ and μ are the weights and biases of the network, F_{ANN} is the approximation and F is the desired output value.

Code Optimization

The search for better results using the proposed artificial neural network may require a large number of neurons in the intermediate layers or even a large number of examples to be used to train the network, consequently demanding a huge amount of processing time. Therefore, performance optimization of the code is a requirement.

Timing and profiling of the sequential code was done in order to identify performance bottlenecks (Stephany et al, 2000). This provided a path for code optimization and further parallelization.

Classical, hardware-independent optimizations were performed in the sequential code and this led to a reduction of 40% in the processing time for the particular neural network architecture of two intermediate layers each with three neurons in the Lorenz's test case. The input layer consists of six neurons while the output layer consist of three neurons. The input layer neurons correspond to forecast and observation data of a given parameter whereas the output layer neurons correspond to the assimilated data. The network was trained using 2,000 examples. The original Fortran 77 code was automatically generated using the MATLAB programming environment, which showed inefficient due to excessive number of calls to subroutines. The Fortran 77 code was compiled using a Fortran 90 compiler in order to use the intrinsic functions of the latter. This allowed replacement of subroutines of nesting level two by calls to Fortran 90 functions, as in the case of loops implementing the dot product of lines of a matrix by columns of another matrix. This procedure was also applied to the chosen transfer function, the hyperbolic tangent, available as the Fortran 90 *tanh* function. The use of programming language intrinsic functions helped to eliminate inefficient hand-coded subroutines, as they are usually well optimized. An alternative would be to use a Fortran 77 compiler, linking to a Mathematics library, that implements, for instance, linear algebra subroutines.

The optimized code was parallelized using HPF-High Performance Fortran (High Performance Forum, 1993) directives, mainly the *INDEPENDENT* statement, specifying that there are no data dependencies between loop iterations and the *FORALL* statement, fully replacing the classical iteration-structured loop. A further version, using Message Passing Interface (MPI) library calls was also implemented, in order to compare both approaches. The MPI directives were mainly imbedded in the routine that determines internal activation of the network as well as the application of the tangent sigmoid function.

Tests for the data assimilation process were performed on the Lorenz's system, given by following dynamical equations:

$$\frac{dX}{dt} = -\sigma(X - Y) \tag{3}$$

$$\frac{dY}{dt} = RX - Y - XZ \tag{4}$$

$$\frac{dZ}{dt} = XY - bZ \tag{5}$$

this system is integrated using the Euler predictor-corrector method adopting $\Delta t=0.001$, $\sigma=10$, b=8/3, R=28 so that the system is in chaotic state, with initial conditions

$w \equiv [X\ Y\ Z]^T = [1.508870 \quad -1.531271 \quad 25.46091]^T$.

The numerical experiment was made inserting *observations* every 12 time-steps. The observational data were the same as forecast data added to a Gaussian deviations with zero-mean. This is the same test carried out by (Miller et al, 1994). For the data assimilation procedure tests were also performed with 10 neurons in the two hidden layers. In the case of 3 neurons weights and bias were determined after 259 iterations whereas for 10 neurons iterations were 152. However, the quality of assimilation was poor when using 3 neurons in the hidden layers. This case is shown in Figure 2. A perfect assimilation was obtained with 10 neurons as shown in Figure 3.

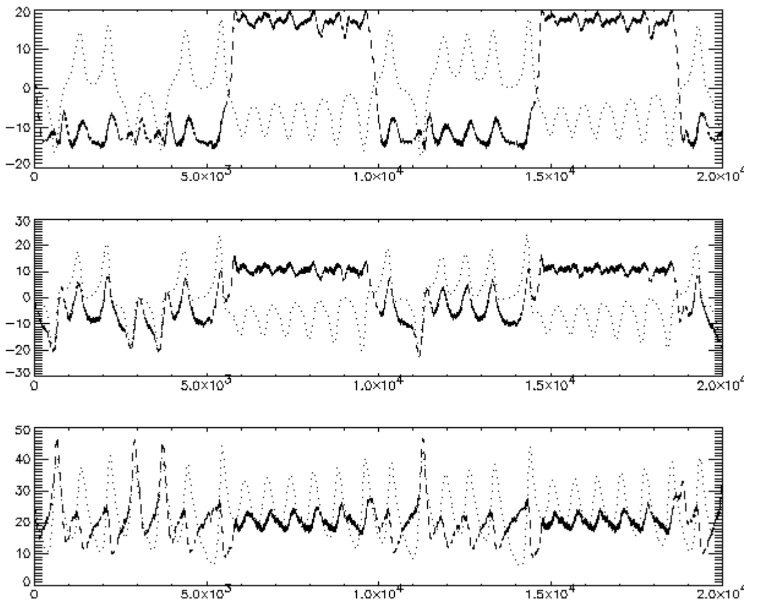

Figure 2. Lorenz Model - Assimilation with 3 neurons in the hidden layers

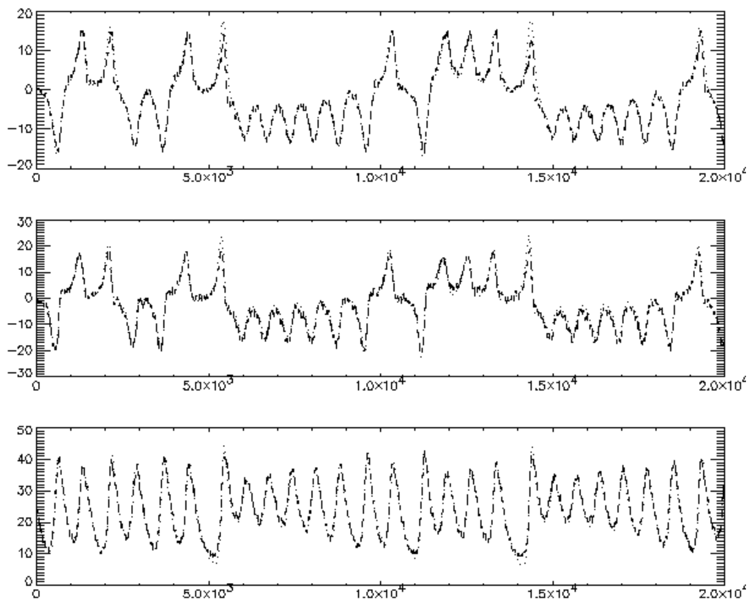

Figure 3. Lorenz Model - Assimilation with 10 neurons in the hidden layers

Tests for data assimilation were also performed on the system based on shallow water equation [Lynch, 1984]. Dynamical equations for this model are shown:

$$\frac{\partial \zeta}{\partial t} + R_o \frac{\partial (u\zeta)}{\partial x} + \delta + R_\beta v = 0 \tag{6}$$

$$\frac{\partial \delta}{\partial t} + R_o \frac{\partial (u\delta)}{\partial x} - \zeta + R_\beta u + \frac{\partial^2 \phi}{\partial x^2} = 0 \tag{7}$$

$$\frac{\partial \phi}{\partial t} + R_o \frac{\partial (u\phi)}{\partial x} - R_o u_0 v + R_F \delta = 0 \tag{8}$$

where u, v are zonal and meridional wind components; ϕ is the geopotential; $\delta = \partial u/\partial x$ is the divengence; $\zeta = \partial v/\partial x$ is the vorticity; R_o=0.10, R_F=0.16, R_β=10, are dimensionless numbers: Rossby, Froude, and a number that gives the importance of β-effect (Lynch, 1984). Hereafter prognostic variables will be grouped into a vector $\boldsymbol{w}=[\zeta \;\; \delta \;\; \phi]^T$. The system is discretized using forward and central finite difference method for time and space integration, where $N_x\Delta x=L$=10000 Km, being L the total length of the channel; N_x=32 the number of grid points; and Δt=100 seconds.

The numerical experiment was made inserting *observations* every 11.1 hours. The observational data were the same as forecast data added to a Gaussian deviations with zero-mean. For the data assimilation procedure tests were performed with 50 neurons in the two hidden layers. The weights and bias were generated after 179519 iterations and the program took several days to finish its execution. Figures 4, 5 and 6 show the assimilation for the geopotential, u component and v components respectively.

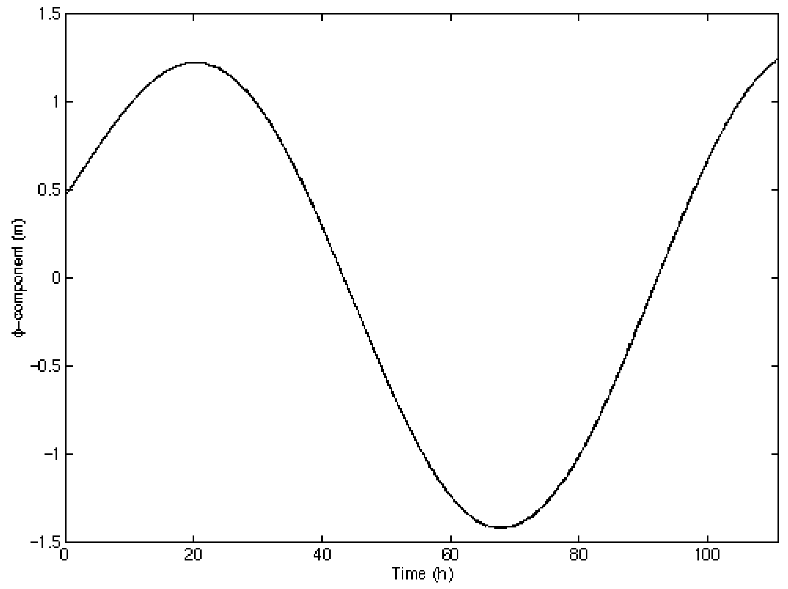

Figure 4. Shallow water equation model - Assimilation with 50 neurons in the hidden layers (Geopotential)

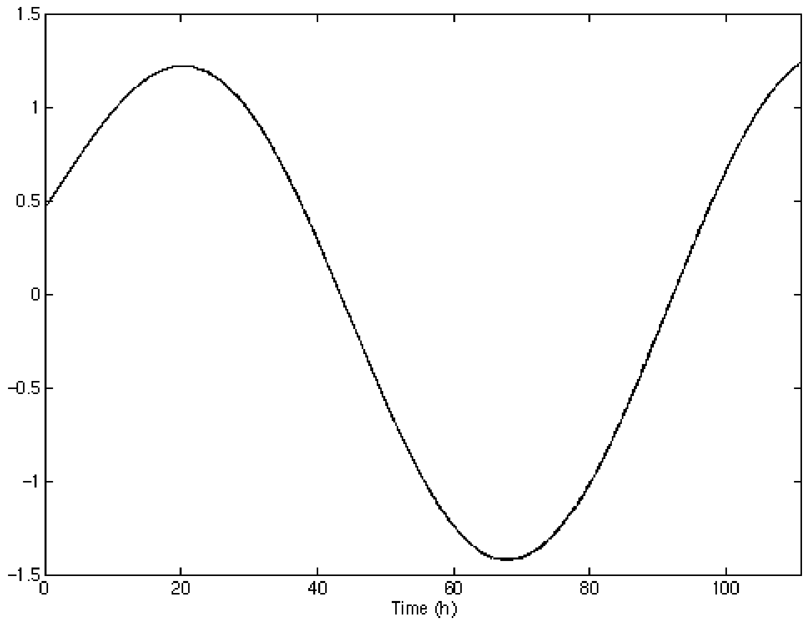

Figure 5. Shallow water equation model - Assimilation with 50 neurons in the hidden layers (U component)

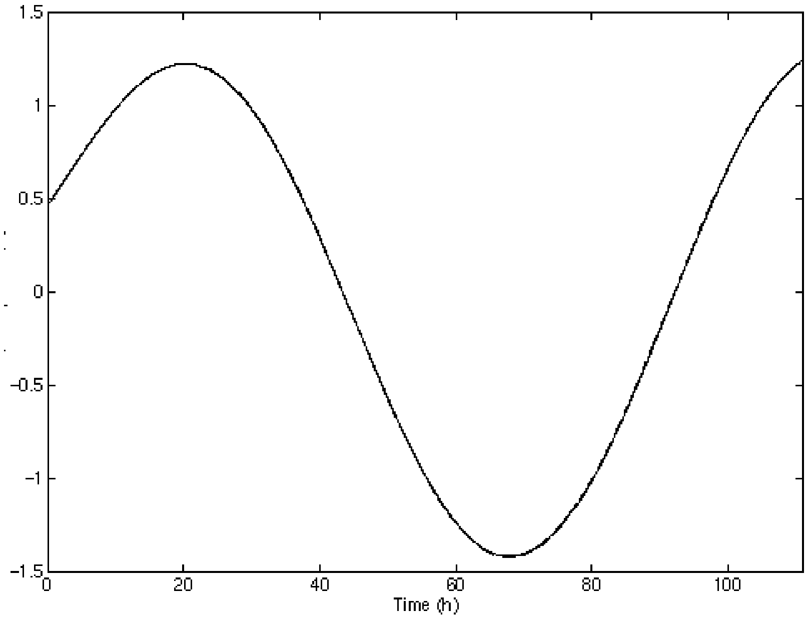

Figure 6. Shallow water equation model - Assimilation with 50 neurons in the hidden layers (V component)

The parallel code is being currently ported to a distributed memory parallel machine. Tests will be conducted to evaluate the efficiency of the parallel implementation in both the HPF and MPI versions. Some preliminary studies have showed that the communication between processors needs to be further optimized. Therefore, the code is being instrumented with timing directives in order to determine precisely the communication bottlenecks in both implementations. It is foreseen that some modifications in the code must be introduced in order to overlap communication and processing.

Final Remarks

The use of the proposed neural network for Data Assimilation proved to be a good alternative as the results confirmed its feasibility. In the case of Lorenz model, convergence was not possible with two neurons but with three. The results of assimilation were improved pretty much when using ten neurons in the intermediate layers. For shallow water equation model, convergence was possible only after using fifty neurons in the intermediate layers. It is impossible to provide a recipe not only with the appropriate number of intermediate layers but also the number of neurons to be used in these layers with this neural network topology. It is merely a question of numerical and experimentation basis. Due to the times involved in processing, optimization and parallelization is very much justified. Optimization made a significant progress in decreasing the processing time.

Work is in progress to imbed HPF directives as well as Message Passing Interface library calls into the optimized code. Tests are being performed in a IA-32 16-cluster environment.

Future work points to the use of other transfer functions such as logistic sigmoid, use of other network topologies such as radial base function.

References

A.F. Bennet (1992) **Inverse Methods in Physical Oceanography**. Cambridge University Press, Cambridge, EUA.

L. Bengtsson, M. Ghil, E. Kälen (1991) **Dynamic Meteorology: Data Assimilation Methods**. Springer-Verlag, New York, EUA.

R. Daley (1991) **Atmospheric Data Analysis**. Cambridge University Press, Cambridge, EUA.

M.W. Gardner, S.R. Dorling (1998): Artificial Neural Networks (The Multilayer Perpectron) - A Review of Applications in the Atmospheric Sciences. *Atmospheric Environment*, **32(14/15)**, pp. 2627-2636.

S. Haykin (1994**) Neural Networks: A Comprehensive Foundation**. Macmillan, New York.

High Performance Fortran Forum (1993) High Performance Fortran Language specification version 1.0. *Technical Report CRPC-TR92225*, Center for Research on Parallel Computation, Rice University, Houston, USA.

E.N. Lorenz (1963): Deterministic nonperiodic flow. *Journal of Atmospheric Science*, **20**, 130-141.

P. Lynch (1984) DYNAMO: A One-Dimensional Primitive Equation Model. *Technical Note 44*, Irish Meteorological Service, Ireland.

R.N. Miller, M. Ghil, F. Gauthiez (1994) Advanced Data Assimilation in Strongly Nonlinear Dynamical Systems. *Journal of the Atmospheric Sciences*, **51(8)**, pp.1037-1056.

A.G. Nowosad, A. Rios Neto, H.F. Campos Velho (2000a) Data Assimilation in Chaotic Dynamics Using Neural Networks. *Proceedings of Third International Conference on Nonlinear Dynamics, Chaos, Control and Their Applications in Engineering Sciences*, July 31 - August 4, Campos do Jordão (SP), Brasil, Vol. 6, Chapter 6 - Control, pp.212-221.

A.G. Nowosad, H.F. Campos Velho, A. Rios Neto (2000b*) Neural Network as a New Approach for Data Assimilation*. Proceedings of the XI Brazilian Congress on Meteorology, pp.3078-3086, Rio de Janeiro, Brazil.

P. Pacheco (1996) **Parallel Programming with MPI**. Morgan Kaufmann Publishers.

S. Stephany, R.V. Correa, C.L. Mendes, A.J. Preto (2000) Identifying performance bottlenecks in a radiative transfer application. **In**: *Applications of High-Performance Computers in Engineering*, Edited by M. Ingber, H. Power, C.A. Brebbia, WIT Press, Southampton.

Todling R. (1997) Estimation Theory and Foundations of Data Assimilation. *Course Notes*, LNCC, Rio de Janeiro, Brazil.

S. Yang, W.R. Cotton (1998) A Method of Continous Data Assimilation using Short-range 4D-Var Analysis. *Technical Report - paper No. 653*, Department of Atmospheric Science, Colorado State University, Fort Collins (CO), USA.

P. Zannetti (1990) **Air Pollution Modeling**. Computational Mechanics Publications, UK.