Program Generation Techniques for the Development and Maintenance of Numerical Weather Forecast Grid Models

Victor V. Goldman[1] and Gerard Cats[2]

[1] Dept. of Computer Science, University of Twente,
PB 217, 7500 AE Enschede, The Netherlands
[2] Royal Netherlands Meteorological Institute,
PB 201, 3730 AE De Bilt, The Netherlands

Abstract. This article presents computer-algebra based techniques for the automatic generation and maintenance of numerical codes based on finite difference approximations. The various generation phases – specification, discretization, implementation and translation – as well as their respective knowledge bases, are discussed and specific attention is given to data mappings in the implementation phase and to high-performance language extensions in the Fortran translation phase. The generation of Fortran source for the dynamics part of a limited area weather forecasting grid-point model is discussed and is illustrated by showing the production of a few variants of the surface-pressure tendency code using the present prototype. Finally, we indicate briefly how adjoints can be obtained using the present methodology.

1 Introduction

A common problem facing efficient maintenance of large scientific codes is the necessity of recoding for porting to new parallel architectures. Although one can nowadays appeal to a variety of tools to assist in such a task, most of them can usually be characterized as source-transforming packages which either compile directly or produce new source appropriate for the target architecture and compiler. Converters such as CMAX [16] fall into the latter category. Experience shows however [19], that most tools do not perform satisfactorily without additional manual intervention or additional automated preprocessing [6], especially for porting from vector to MPP architectures. Although the standardization of data parallel languages such as HPF is expected to significantly improve portability, many proposed automatic data layout and distribution tools assume coded Fortran source as starting point. In the present context, a difficulty with automatic source transformation is that important knowledge which is used in developing the code is not available to the automatic tool at the coded program stage. This knowledge is often useful for attaining an acceptable level of optimality. The problem is usually compounded when extensibility (e.g. amending or extending the algorithm) is added to the portability issue.

In this paper we adopt a program generation methodology to address such maintenance issues. The generation process, which is partially rule-driven, starts from a compact equational specification. After a number of transformations which include discretization and data mapping, the generator produces source code: Fortran 77 or 90, optionally with some high-performance constructs. It also generates code using memory mappings suitable for vector as well as for MPP architectures. The programming platform of the generation is the Lisp-based computer algebra system REDUCE [7] and the translation of intermediate code makes use of the REDUCE-based GENTRAN [4] package for Fortran 77 and GENTRAN 90 [2], its Fortran 90 extension. To demonstrate the potential of the present methodology, it was applied to the equations that are solved by the HIRLAM [3] model. This is a numerical weather forecasting model that is in operational use in most of the meteorological institutes of the countries that participate in the HIRLAM project.

Because of the time constraints on weather forecast production it is essential that the HIRLAM model code is efficient on the available hardware. Therefore models like this pose a real challenge on automatic code generation systems with respect to code optimality. Currently, however, we accept that the generation of optimal code is aided by the manual specification of a set of heuristic rules to the generator. Our research is more directed towards the generation of (Fortran) codes for a range of models from a basic set of equations: The model proper, its linearized version and the adjoint of the latter. A reason to choose HIRLAM in particular is that currently within the HIRLAM project group a task force is operating to develop a linearized version of the HIRLAM model and its adjoint. For sake of brevity, in this paper we will restrict ourselves to the generation of the equation for surface pressure tendency as solved by HIRLAM and its linearization and adjoint. Within this research, we will assume that the model equations are given in discretized form. The derivation of the discretized equations from the continuous forms is a task for numerical experts where other constraints than the model equations (e.g. conservation of energy) ([1]) play a role. Here we do not attempt to build an expert system to automate that process.

The HIRLAM forecast model and the discretized pressure tendency in particular are explained in Sect. 2. In Sect. 3 the structure of the prototype generator is described. The procedure for generating surface-pressure tendency code is presented in the Sect. 4. Finally in Sect. 5, we briefly describe automatic generation adjoints for data assimilation using the present methodology.

2 The Discretized Forecast Model

Overview. The main components of the HIRLAM forecasting system consist of a data assimilation scheme, to construct the initial state of the atmosphere from observations, and the forecast model, to integrate the atmospheric equations of

[3] The HIRLAM model was developed by the HIRLAM project group, a cooperative project of Denmark, Finland, Iceland, Ireland, The Netherlands, Norway and Sweden.

motions in time. The current formulations of these components allow efficient implementation of the forecast model and its adjoint on massively parallel systems, but not of the data assimilation scheme. This is one of the reasons that currently efforts are directed towards reformulation of the latter scheme into one based on variational techniques using the adjoint equation of the forecast model ([15]). Therefore, it is not a serious restriction that we will limit ourselves here to the forecast model and its adjoint.

The forecast model consists essentially of two main parts. The first, called 'dynamics', solves the primitive equations of motion, e.g., conservation of mass, momentum, and energy, by time and space discretization techniques. The second, called 'physics', describes the effect of sub-grid scale processes on the discretized model variables. It is a characteristic of the division into physics and dynamics that the former do not contain horizontal exchange of information: Physics are formulated column-wise, all horizontal 'communications' take place within the dynamics part. In general, the dynamics form the challenging part for code generation systems because physics are embarrassingly parallel under horizontal domain decomposition (which is the usual vectorization and parallelization strategy, [3]). For this study we selected from the dynamics the equation describing conservation of mass, because it is representative for most dynamical equations, yet conceptually simple. A class of equations we do not address here consists of those implying 'global communications', like Helmholtz equations or Fourier transforms. The following is thus restricted to the grid-point version of the HIRLAM model.

The Surface-Pressure Tendency Equation. In its vertically integrated form, conservation of mass reads:

$$\frac{\partial p_s}{\partial t} = -\int_0^1 \nabla \cdot \left(\mathbf{v}_h \frac{\partial p}{\partial \eta} \right) d\eta \tag{1}$$

In here, p_s is surface pressure, t time, $\nabla\cdot$ the divergence operator, p pressure, and $\mathbf{v}_h$ the horizontal wind. The vertical coordinate is η ([13]). Equation (1) is the pressure tendency equation.

The vertical discretization is performed by defining the pressure at NLEV + 1 so-called half-levels by two sets of numbers, A and B:

$$p_{k+1/2} = A_{k+1/2} + B_{k+1/2} p_s(x, y) \ ; \ k = 0, \ldots, \text{NLEV} \tag{2}$$

Wind components u and v are given on NLEV 'full-levels', at which the pressure is the arithmetic average of the pressures at the two neighboring half-levels. The surface is half-level NLEV + 1/2, so $A_{\text{NLEV}+1/2} = 0$ and $B_{\text{NLEV}+1/2} = 1$.

The horizontal layout of the grid points is the Arakawa-C grid ([1]) where the wind components u and v are given half a grid distance to the East and North, resp., with respect to surface pressure p_s. (Fig. 1). The surface pressure tendency equation (1) can be discretized in a variety of ways. In combination with suitable discretizations of the other model equations, the following form satisfies the additional constraints like conservation of energy:

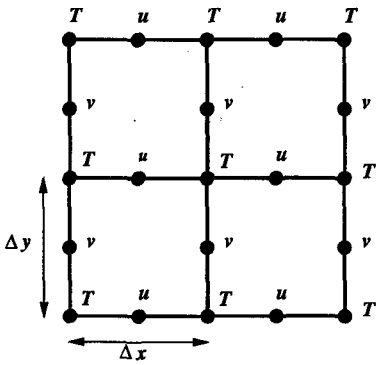

Fig. 1. The Arakawa-C grid. The figure shows the position of the grid points of wind components (u and v) with respect to the main grid points (T), carrying all other prognostic model variables

Define:

$$U_k = \overline{\Delta p_k}^{x} u_k \ ; \ V_k = \overline{\Delta p_k}^{y} v_k \tag{3}$$

The overbar denotes the operator to obtain the value at a grid point where there is no directly available value; simple arithmetic averaging is sufficiently accurate (second order in Δx). Further symbols to be used are: a: earth radius; h_x and h_y: map factors (to describe e.g. convergence of meridians towards the poles); Δ: vertical difference; δ: horizontal difference. Then:

$$\begin{aligned} \frac{\partial p_s}{\partial t} &= -\sum_{j=1}^{\mathrm{NLEV}} \frac{1}{a h_x h_y} \{\delta_x(h_y U_j) + \delta_y(h_x V_j)\} \\ &= -\frac{1}{a h_x h_y} \left\{ \delta_x \left(h_y \sum_{j=1}^{\mathrm{NLEV}} U_j \right) + \delta_y \left(h_x \sum_{j=1}^{\mathrm{NLEV}} V_j \right) \right\} \end{aligned} \tag{4}$$

Equations (3) and (4) are the equations we will consider in the sequel.

3 The Generation Process

Overview. Program generation, or more broadly, *software synthesis* [12], is a branch of software engineering which concerns itself with the automation of program writing. In a knowledge-driven approach, coded knowledge is used by the generator to transform a compact specification into a ready to compile and execute program. An important aspect of knowledge-based software synthesis is that once represented and coded, the knowledge can be easily reused, amended, or extended, thus facilitating maintenance. In the field of computational science, there is a great deal of research being done on the many issues in this area: from computational intelligence, knowledge bases, and environments (see e.g.

[8, 18, 10]), to expression manipulation, optimization and translation (see e.g. [9, 4, 2]). Many software generators [17, 10, 18] make use of computer algebra systems, especially to manipulate mathematical expressions. The present generator is embedded in the computer algebra system REDUCE [7]. The overall architecture of the generator is depicted in Fig. 2. The specifications contain computational *scripts* which delineate concisely the computational trajectory. High-level optimization heuristics are also part of scripts, but routine low-level optimizations are implicitly left to the compiler.

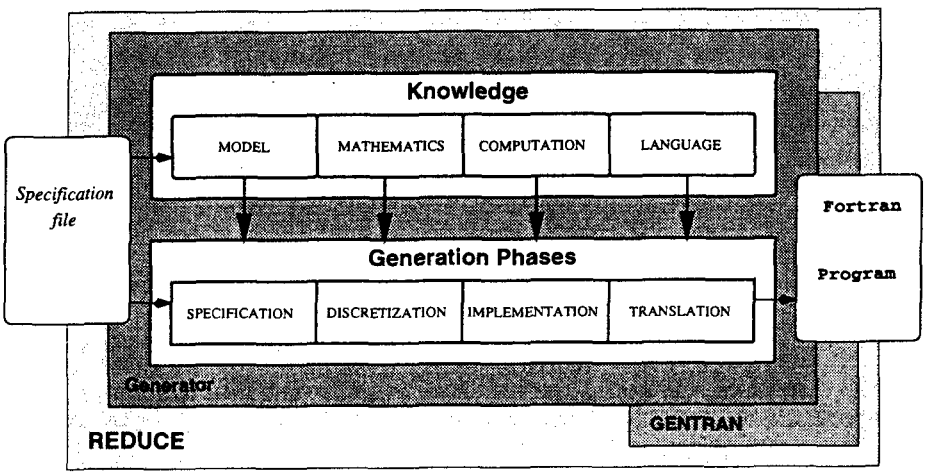

Fig. 2. General architecture of the generator

Knowledge Sources. On an abstract level there are essentially four knowledge sources: *model knowledge*, *mathematical knowledge*, *computational implementation knowledge*, and *target language knowledge*. A great deal of this knowledge is implemented in the form of equations and rules. When invoked by an object, a set of REDUCE rules will recursively perform substitutions very much like rewrite systems. The knowledge categories are not fully independent of each other: knowledge in one category is sometimes expressed in terms of knowledge of another.

Model knowledge includes model equations such as those discussed in Sect. 2 and forms the basis for constructing *scripts*. Model discretization also belongs to this category. Mathematical knowledge embodies the necessary mathematical machinery. Besides implicit knowledge contained in the computer algebra system, it includes definitions of finite difference operators and approximations for differential and integral operators. Computational implementation knowledge is used in transforming the discretized equations into a computation on a computational grid. Information concerning program variables, their memory mappings

and shapes is also contained here. Finally, translation knowledge embodies target language syntax and target architecture knowledge, the former being mostly contained in the GENTRAN [4] and GENTRAN 90 [2] packages.

Generation Phases. Generation proceeds through *specification*, *approximation-discretization*, *implementation* and *translation* phases. They are executed procedurally, drawing from the declarative knowledge sources which were summarized above. Besides scripts, variable declarations and data mapping directives form important components of the specifications. Knowledge can also be updated in the specifications. In the context of data parallel computing, script boundaries should correspond as much as possible to program phase[4] boundaries. In the discretization phase, differential and integral operators acquire their discretized form and, after all finite differencing operators are evaluated, all field variables appearing in the script equations are placed on the (staggered) discretization grid. An inconsistent grid-point reference will be detected.

The implementation phase transforms the discretized scripts into an imperative calculation. Indexed variables are now associated with arrays; staggered variables get integer array indices. Some of the program transformations include index mappings, array reduction and array declarations, etc. The abstract computation is now expressed as intermediate code and passed to the translation phase. The latter is performed by the GENTRAN packages.

4 The Surface-Pressure Tendency Code Generation

Specification and Scripts. We illustrate below a portion of the specification for the pressure tendency code. It is written in the syntax of REDUCE and its associated support language Rlisp. Certain entries such as grid definition and associated field variables declarations are not shown. Another script defining temporary variables to improve efficiency through common subexpression elimination, exchanging divisions for multiplications, as well as other heuristics is also omitted. Equation (2) for p and equation (3) for U and V are represented through the assignments of the generator variables `p_r`, `uaux_r` and `vaux_r` respectively. The operator `df` is the REDUCE symbolic differentiation operator. The discretization rules determine its finite difference approximation. The operator `centr_av` is a central average operator, and `div2_op` represents the horizontal divergence expressed in curvilinear coordinates.

[4] The word *phase* is overloaded in this article. When the generator itself is discussed, the conventional meaning of the word is intended. When referring to generated programs, as is done here, the word takes its more recently acquired technical designation: a program segment with specific array reference characteristics, (see e.g. [11]). Confusion of this type is nearly unavoidable when discussing programs which generate programs. Other program attributes such as *implementation* suffer also to some extent from the same ambiguity.

```
% ----------------------------------------------------------------
%                    Pressure Tendency Script
% ----------------------------------------------------------------
p_r:=A+B*ps                                                      $
    da_eq:=     d_a=df(a,eta)                          *   d_eta$
    db_eq:=     d_b=df(b,eta)                          *   d_eta$
    dp_eq:=     d_p=df(p_r,eta)                        *   d_eta$
    dp_eq:=     (dp_eq where
                  heuristics_rules0({da_eq,db_eq}))              $

uaux_r:=u*centr_av(d_p,x)                                 *2/d_eta$
vaux_r:=v*centr_av(d_p,y)                                 *2/d_eta$

    u_aux_eq:=  u_aux=numeric_integral(uaux_r,eta,0,1)           $
    v_aux_eq:=  v_aux=numeric_integral(vaux_r,eta,0,1)           $

        repla_unknown!*:='((hx.hxv) (hy.hyu))                    $

eq_4:=          ddt_ps= -div2_op(u_aux,v_aux)                /  2$
ps_tendency_script:={da_eq,db_eq,dp_eq,u_aux_eq,v_aux_eq,eq_4}   $
% ----------------------------------------------------------------
```

Embodied in the REDUCE program variable `ps_tendency_script` (last program line), is the actual definition of the script used to generate the computation of (4). It is a concise list of equations (also defined in the specification) which delineate the steps of the computation on a high level. It was obtained from the second equality in (4). The list of equations could be modified, e.g. to represent the first equality in (4), should that lead to more efficient implementations on available hardware architectures. The present breakdown corresponds roughly to the one in the current (Fortran 77) reference version of the HIRLAM model, chosen because it leads to a more efficient code for vector architectures than the first equality. A posteriori it became apparent that a small number of factorizations would further improve efficiency. They appear in the script flushed to the right for the sake of clarity.

Data Mappings. The storage association of the field variables is determined by specifying index transformation functions. The corresponding inverse transformations are also needed because one is dealing with the mapping of functions and not of index values. A mapping of the horizontal index set (m, n) onto a column-major storage index set (ii) can be specified by the REDUCE expression:

```
vec_map := { {ii= m-low(m)+1+(n-low(n))*npts(m)},
             {n=floor((ii-1)/npts(m))+low(n),
              m=(ii-1)-floor((ii-1)/npts(m))*npts(m)+low(m)} };
```

The second and third lines represent the inverse map. Presently a mapping is applied globally to all field quantities, but future implementations will enable assigning a specific mapping to a specific variable as well as to a specific program phase.

Translation. Omitting the automatically generated array declarations, as well as temporary variables code, the program segment generated from the above script and mapping is shown below. The loop ranges were obtained via an iter-

```
      DO 25012 KK=1,MLEV                                    DO 25019 II=1,MLAT*MLON-MLON
      ZDAK(KK)=AHYB(KK+1)-AHYB(KK)                          V_AUX(II)=0.0
25012 CONTINUE                                        25019 CONTINUE
      DO 25013 KK=1,MLEV                                    DO 25020 KK=1,MLEV
      ZDBK(KK)=BHYB(KK+1)-BHYB(KK)                          DO 25021 II=1,MLAT*MLON-MLON
25013 CONTINUE                                              V_AUX(II)=PVZ(II,KK)*ZDPK(II+MLON,KK)+
      DO 25014 KK=1,MLEV                                   . PVZ(II,KK)*ZDPK(II,KK)+V_AUX(II)
      DO 25015 II=1,MLAT*MLON                         25021 CONTINUE
      ZDPK(II,KK)=PPSZ(II)*ZDBK(KK)+ZDAK(KK)          25020 CONTINUE
25015 CONTINUE                                              DO 25022 II=MLON+2,MLAT*MLON-MLON-1
25014 CONTINUE                                              PDPSDT(II)=ZRDLOH*HYU(II-1)*U_AUX(II-1)
      DO 25016 II=1,MLAT*MLON-1                            . *ZRHXHY(II)-(ZRDLOH*HYU(II)*U_AUX(II)*
      U_AUX(II)=0.0                                        . ZRHXHY(II))+ZRDLAH*HXV(II-MLON)*V_AUX(
25016 CONTINUE                                             . II-MLON)*ZRHXHY(II)-(ZRDLAH*HXV(II)*
      DO 25017 KK=1,MLEV                                   . V_AUX(II)*ZRHXHY(II))
      DO 25018 II=1,MLAT*MLON-1                       25022 CONTINUE
      U_AUX(II)=PUZ(II,KK)*ZDPK(II+1,KK)+PUZ(
     . II,KK)*ZDPK(II,KK)+U_AUX(II)
25018 CONTINUE
25017 CONTINUE
```

ated DEFINE/USE consistency scheme as a way of treating boundaries. Two iterations were needed. Certain optimizing transformations such as factorizations, loop-fusion, and code motion will improve the code further. Implementation of such transformations is in progress.

Presently, maintenance of forecast models involves porting code with two-dimensional arrays for vector machines to fully three-dimensional arrays for MPP architectures. By specifying a three-dimensional array storage scheme instead of the one above, a similar code is generated with the correct array references and three-level loop nesting. We show instead below the elemental assignment version with **eoshift** and **sum** operators in Fortran 90 syntax using GENTRAN 90.

```
 real,dimension(mlev+1)::ahyb,bhyb,zdak,zdbk
 real,dimension(mlon,mlat)::ppsz,pdpsdt,zhx,zhy,hxv,hyu,u_aux,v_aux&
& ,zrhxhy,hxhy,rhxu,rhyv
 real,dimension(mlon,mlat,mlev+1)::zpkp,zdpk,puz,pvz,ke

 zdak=eoshift(ahyb,1,1)-ahyb
 zdbk=eoshift(bhyb,1,1)-bhyb
 zdpk=ppsz*zdbk+zdak
 u_aux=sum(eoshift(zdpk,1,1)*puz+puz*zdpk,3)
 v_aux=sum(eoshift(zdpk,2,1)*pvz+pvz*zdpk,3)
 pdpsdt=zrdloh*eoshift(hyu,1,-1)*eoshift(u_aux,1,-1)*zrhxhy-zrdloh*&
& hyu*u_aux*zrhxhy+zrdlah*eoshift(hxv,2,-1)*eoshift(v_aux,2,-1)*   &
& zrhxhy-zrdlah*hxv*v_aux*zrhxhy
```

The array declarations as well as the `DIM` and `SHIFT` parameters of `eoshift` were inferred automatically using special generic procedures. We have not yet studied in detail the efficiency of the code in this form. However its compactness makes it more amenable to automatic common subexpression elimination and other transformations, obviating the manual breakdown of the scripts discussed earlier, as shown by our preliminary work using the SCOPE [9] package.

5 Adjoint Code Generation

The maintenance of large model codes is usually accompanied by the maintenance of the adjoint model. The adjoint has been traditionally hand-coded but more recently tools have been constructed for its automatic generation through the use of *automatic differentiation* techniques (see e.g. [5] for an overview). Like many automatic tools, automatic adjoint generators take (hand-)coded source as their starting point. As already implied in the introduction, source code often contains programming constructs which are more relevant to the target language and hardware architecture. These usually mask model and mathematical knowledge which is needed to perform what is in principle a mathematical transformation. The generation techniques presented here allow us to couple the generation of the adjoint model to the generation of the forward model with relative ease. Within the framework shown in Fig. 2, the adjoint generation is initiated at the specification/discretization level.

In adjoint modeling it is the adjoint $\mathcal{D}^*$ of the linearized form of the forward model D which is needed. The action of $\mathcal{D}^*$ on a vector a in the range space of D can be expressed in terms of the inner product (a, Du), as:

$$\mathcal{D}^* a = \frac{\delta(a, Du)}{\delta u} \quad , \tag{5}$$

as discussed by Thacker [14] in the context of automatic differentiation. The adjoint code can thus be generated by differentiating the appropriately discretized inner product. In terms of specification scripts, we introduce the variable `adj_ddt_ps` adjoint to the surface pressure tendency. Its contribution to the inner product for the dynamics will be the horizontal sum of `adj_ddt_ps*ddt_ps`. By prepending the script of the forward model one obtains a full script for the inner product. An adjoint script `adj_script` is then generated by performing differentiation on `inner_prod_script` in reverse mode with respect to the model variables `u`, `v`, and `ps`. The scripts for the inner product as well as for the adjoint are shown below. The latter was automatically generated from the former, and has the recursive structure typical of reverse mode differentiation. The last three equations in the script represent the required adjoints of `ps`, `u` and `v` respectively. Fortran code generation follows by going through the same transformations as for the forward model. Details of this work, which is in progress, will be presented elsewhere.

```
inner_prod_script:=
   {ps_tendency_script,

  inner_prod=
    discr_integral(discr_integral(
    adj_ddt_ps*ddt_ps,
   x,0,rmaxlon),y,0,rmaxlat)}$


mod_vars!*:='( ps u v)$
mod_range_vars!*:='( ddt_ps)$
```

```
adj_script :=
  {adj_a1001=df(inner_prod,ddt_ps'),
   adj_b1001=mat_mul(adj_a1001,df(ddt_ps,u_aux')),
   adj_b1002=mat_mul(adj_a1001,df(ddt_ps,v_aux')),
   adj_c1001=mat_mul(adj_b1001,df(u_aux,d_p')) +
             mat_mul(adj_b1002,df(v_aux,d_p')),
   adj_c1002=mat_mul(adj_b1001,df(u_aux,u')),
   adj_c1003=mat_mul(adj_b1002,df(v_aux,v')),
   adj_d1001=mat_mul(adj_c1001,df(d_p,ps')),
   adj_d1002=mat_mul(adj_c1002,df(u,u')),
   adj_d1003=mat_mul(adj_c1003,df(v,v'))}          $
```

6 Conclusions

We have presented program generation techniques for large scale numerical models. We have concentrated more on procedural components than on high-level inference in order to be able to assess the functionality needed to be able to obtain the necessary leverage for realistic maintenance and development issues. In spirit, our work is mostly related to that of Ref. [17] and, to some extent, to that of Ref. [10] where the architecture has much in common with ours. That work however has a much more developed inference system and more formalized specifications. Specifications in the present work are less constrained so as to allow us to deal more directly with the issues mentioned in the article. Future work includes generating other parts of the forecast model, and implementing additional code transformations and more sophisticated data layout schemes.

7 Acknowledgments

The authors have benefited from discussions with Lex Wolters, Robert van Engelen, Paul ten Brummelhuis, and Hans van Hulzen. We are grateful to André Koopal for providing LaTeX forms of the HIRLAM equations.

References

1. Arakawa, A., Lamb, V. R.: A potential enstrophy and energy conserving scheme for the shallow water equations. Mon. Wea. Rev.**109** (1981) 18–36.
2. Borst, W. N., Goldman, V.V., van Hulzen, J. A.: GENTRAN90: A REDUCE package for the generation of FORTRAN 90 code. (ISSAC '94),Proceedings Int. Symp. on Symbolic and Algebraic Computation (1994) 45–51. ACM Press, New York.
3. Cats, G., Middelkoop, H., Streefland, D., Swierstra, D.: In: The dawn of massively parallel processing in meteorology, Springer Verlag, Berlin (1990) 47–75.
4. Gates, B. L., Dewar, M. C.: GENTRAN User's Manual - REDUCE VERSION, (1991)
5. Griewank, A., Corliss, G.F., eds.: *Automatic Differentiation of Algorithms.* SIAM, Philadelphia, (1991).

6. Hammond, S. W., Loft, R. D., Dennis, J. M., Sato, R. K.: A data parallel implementation of the NCAR Community Climate Model (CCM2). In *Proc. Seventh SIAM Conference on Parallel Processing for Scientific Computing*, D.H. Bailey, P.E. Bjørstad, J.R. Gilbert, M.V. Mascagni, R.S. Schreiber, H.D. Simon, V.J. Torczon and L.T. Watson, (eds.), (1995) 125–130. SIAM, Philadelphia.
7. Hearn, A. C.: REDUCE 3.5 Manual (1993), The Rand Corporation, Santa Monica.
8. Houstis, E. N., Rice, J., R., Vichnevetsky, R., (Eds.): Proceedings of the Third International Conference on Expert Systems for Numerical Computing, West-Lafayette,May 1993. Math. Comput. Simulation **36** (1994) 269–520
9. van Hulzen, J. A.,: SCOPE 1.5, a Source-Code Optimization Package for REDUCE 3.5 – User Manual. Memorandum INF-94-17, Department of Computer Science, University of Twente, (1994).
10. Kant, E., Yau, A. S-H., Liska, R., Steinberg, S.: A problem solving environment for generating certifiably correct programs to solve evolution equations. Preprint (1995). Department of Mathematics and Statistics, University of New Mexico, Albuquerque.
11. Kremer, U., Mellor-Crummey, J., Kennedy, K., Carle, A.: Automatic data layout for distributed-memory machines in the D programming environment. Technical Report CRPC-TR93298-S, Center for Research on Parallel Computation, (1993).
12. Setliff, D., Kant, E., Cain, T.: Practical Software Synthesis. IEEE Software **10** (1993) 6–10
13. Simmons, A., Burridge, D. M.: An energy and angular-momentum conserving vertical finite-difference scheme and hybrid vertical coordinates. Mon. Wea. Rev.**109** (1981) 758–766.
14. Thacker, W. C.: Automatic differentiation from an oceanographer's perspective. In [5], pp. 191–201.
15. Thépaut, J.N., Courtier, P.: Four-dimensional variational data assimilation using the adjoin of a multilevel primitive-equation model. Quart. J. Roy. Meteor. Soc. **117** (1991) 1225-1254.
16. Thinking Machines Corporation: *Using the CMAX converter*. Manual version 2.0 (1994).
17. Wang, P. S.: FINGER: A Symbolic System for Automatic Generation of Numerical Programs in Finite Element Analysis, J. Symb. Comp. **2** (1986) 305–316
18. Weerawarana, S., Houstis, E. N., Rice, J. R.: An interactive symbolic-numeric interface to Parallel ELLPACK for building general PDE Solvers. In B.R. Donald, D. Kapur, and J. L. Mundy, editors: *Symbolic and Numerical Computation for Artificial Intelligence*, Academic Press (1992) 303–321.
19. Wolters, L., Cats, G., Gustafsson, N.: Limited area numerical weather forecasting on a massively parallel computer. *Proceedings of the 8th ACM International Conference on Supercomputing*, July 11-15 1994, Manchester, England, ACM press, (1994),289-296.