

A RETARGETTABLE PROGRAMMING ENVIRONMENT FOR STUDYING CLIMATE MODELS

D. ABRAMSON¹, M. DIX², R. FRANCIS¹, I. MATHIESON¹ and P. WHITING¹

¹*Division of Information Technology and* ²*Division of Atmospheric Research, CSIRO.*

1. INTRODUCTION

Numerical weather prediction models have been used for some years to help provide short range predictions of local weather patterns, and are now being used to provide longer range global climate patterns. Whilst early programs only modelled wind dynamics, modern codes attempt to model physical effects such as solar radiation, clouds, rain, snow and ice. The proposed new generation of programs will also include more detailed physical models such as evaporation through vegetation and will be extended to include coupled ocean effects. The increasing complexity of the models raises two important issues in computer science: what is the most effective way to build large complex weather prediction programs, and how can these programs take advantage of the growing number and diversity of high powered parallel processors?

An example 9 layer climate model in use at the CSIRO Division of Atmospheric Research consists of 20,000 lines of Fortran-77 code. To transport this program to a range of different parallel processors implies substantial recoding and will probably involve the programming of a number of versions tuned to each architecture. For instance, re-coding the program for an array of transputers presents quite different problems to re-coding it for a massively parallel SIMD machine like the Connection Machine. The structures for vector parallelism provided in Fortran-90 will do little to alleviate the problem. A further disadvantage of the current implementation is that the code is quite removed from the original mathematical model on which it is based, and is therefore difficult to maintain. Long term model maintenance is an essential component of experimentation with new physics effects, and thus prototyping new programs is difficult and error prone.

This paper discusses the design of a programming environment, called DWARF, designed to assist in these two areas of developing and maintaining climate modelling programs. DWARF provides a specification language which allows a mathematician to express a climate model with little regard to the underlying machine. Separate code generation phases can be tailored to a number of existing uni-processors as well as many different parallel machines. This environment allows rapid prototyping of new models as well as providing efficient code generation for machines with radically different architectures. Information embodied in the model specification is used to determine an efficient mapping of the problem onto the target architecture. The language is far more specific than a general parallel programming language, and thus, it should be easier and faster to retarget the language to a new machine than conventional compilers. The system has additional advantages by providing model documentation in a form which is readable and which can be automatically included in other research documents.

2. RATIONALE

Climate and weather prediction models are continually evolving. In particular, the parameterization of physical processes is regularly updated, based on new theories or to correct perceived shortcomings in the model simulation. While there are stable versions of the model (in the sense that a particular version is used for a long climate run) there is never a finished product. There are also various *research* versions of the model used by individual scientists to test particular ideas. The changes in these versions may be incorporated in the standard model if they prove successful.

This continual development is fundamental to the science of climate modelling, and demands a high premium be placed on the ability to modify code easily. At present this has usually been sacrificed in favour of optimisations to improve execution speed on the current target machine. The development also means that there cannot be a single, complete specification of the model before it's

built. Many changes can only be decided after long model runs and the documentation and specification of the model can only grow along with the code itself. Because so much of the development is speculative testing of ideas, in some areas there will be little formal specification separate to the code. Such specification is added later if the technique is incorporated into the main model version. While some part of existing model code is the true definition of the model (the basic variables and the numerical methods) a substantial part of the code is essentially extraneous detail or special optimisations. The irrelevant coding structure effectively makes the specification invisible within the program. By developing models in a specification language and having a compiler encode it within the program structures needed by a specific machine, models will be far more accessible to their designers.

A number of parallel languages have been devised in attempts to address the problems cited above. However, most of these are intended as general purpose parallel programming systems, and thus cannot assist in the high level specification of climate models. Further, they do not necessarily produce models that are any easier to maintain or develop than would be the case with their sequential counter-parts. For example, Chang (1990a) translated a spectral global climate model from Fortran to Sisal, a new applicative parallel programming language. Sisal boasts a high degree of machine independence, although to date, most of the implementations are for shared memory multiprocessors. In his study, the Fortran code was 2203 lines, where as the Sisal code was 2888 lines. In a second experiment, he translated a 1588 line sequential Fortran program into a parallel Fortran program using the Encore parallel programming tool EPF (Chang 1990b). In this case the new program was 2047 lines. Thus, the new parallel languages do not necessarily reduce the code size as would be hoped, and maintenance is no simpler although the models may be more portable across different architectures.

The model specification language introduced here provides several benefits over conventional sequential and parallel languages. It separates the specification of the model from the details of the implementation for a particular machine. This should greatly simplify the problem of model development. Any architecture specific optimisations will appear in the code generation module of the compiler. As result, all models will acquire the benefits of them and the model specification will be free of them. Because DWARF is designed to specify a model, the actual code will be more compact than conventional FORTRAN and relate more directly to the mathematics being represented. As an aside, DWARF can also enforce additional kinds of type checking not present in other languages, on the spatial domain and physical dimensions of terms in the model equations.

3. FEATURES OF THE DWARF LANGUAGE

As a computing problem, atmospheric modelling presents the need for maximum floating point performance combined with tremendous regularity in the data structures underlying the calculations and extensive repetition of similar calculations. Initial investigations have highlighted the following aspects of atmosphere modelling:

- The existing CSIRO 9 layer climate model develops $64 * 56 * 9 = 233,856$ cells and new models can be expected to increase this number in all dimensions as well as adding coupled ocean models.
- The data modelling the physical values are written once per time step and intermediate values are not propagated between time steps. The particular time step used in the models involves maintaining three copies of all propagated values.
- The horizontal physical dimensions are very much larger than the vertical physical dimensions and the time scales of the horizontal processes are also very much larger than the vertical processes. Thus models integrate at a time step appropriate to the horizontal scales and averaging approximations are made for vertical processes.

As a consequence of these observations the following assumptions can be made in designing a system to support climate modelling. First, the difference in space and hence time scales in the vertical dimension suggest the use of a two dimensional structure over which the horizontal dimensions should be mapped. Mapping the vertical layer at each horizontal co-ordinate to the same processor will improve the ratio of computational to communication load in the machine. Second, the degree of parallelism in the problem is large and can be expected to grow. For example, horizontal parallelism can readily grow from the existing 3,584 by factors of 4 to 16. Thus massively parallel

solutions are relevant and horizontal spreading offers ample opportunity. Third, the variable writing pattern and time step semantics indicate a functional specification form will be effective for model representation

These design decisions in turn suggest a framework in which weather models can be specified. This framework involves an understanding of the data referencing patterns and access frequencies which occur in atmospheric models and the structural nature of the calculation imposed by the modelling task. The framework can be embodied as a programming language suitable for the high level specification of atmospheric modelling algorithms. The language in turn can be embodied in a compiler capable of converting models into code for a range of parallel and super computers. The overall requirements of this work are threefold. First, the specification language should mirror the problem to the extent that a substantial coding benefit can be obtained by its use. Second, the compiler should generate code from identical source for vector, MIMD and SIMD parallel architectures. Third, the resulting code should execute near the theoretical performance limits of each of the architectures to which it is targeted.

A major consideration in DWARF is the assumption of replicated data structures and the notion of iteration across those spaces. In addition, multiple instances of all data structures exist (representing progress in time) and assignment calculates the values in the next instance of the data from values in the current instance and earlier instances. A model proceeds by establishing initial instances, computing values for the first time step, and then recalculating values for each subsequent time step as specified by the program's control component.

A program written in DWARF consists of a series of constant, domain, function, variable and calculation specifications together with a control component. Constants provide symbolic names for values which can be calculated by the compiler. Domains define the structure within which calculations can occur and control the geometry and hence the implicit replication of that data. For example, a domain could be a rectangular grid and objects defined in that domain are therefore replicated at all grid points. The initial language is restricted to rectangular two and three dimensional domains. Functions support modularisation and have assignment access to the global data of the program. Calculations are specified by assignment where the left hand side is an element of a domain and the right hand side is an expression yielding a value within that domain. The single assignment rule applies. Conditional application within a domain can be specified using the *where* clause over a group of assignment statements. Finally the control component provides for data initialisation, the execution of a first calculation, a general time step and a final calculation.

Major decisions in the language design include the mechanism for specifying replicated calculations, the interaction between domain and *where* clauses, and data declarations and operations, and the specification of communication between elements of a domain and between current and old instances of values. Each of these is discussed in turn in the following sections.

3.1 Domains

Domains provide the fundamental model structuring on which much of the simplification of the programming and code generation is based. A domain defines a region over which data is replicated. Values can only exist at the nodes of a domain and hence calculations which create values are specified for the general node and imply calculation at all nodes. Within a domain, reference to values at remote nodes can be made either through a *transform* or by applying movement operators to any variable. Movement operators are declared as part of the domain specification and essentially represent direction indicators. Thus

$$\text{new } X = (X.n + X.s) / 2.0$$

implies the averaging of the neighbours in the north and south direction to obtain the new value for *X* at every point in the domain over which *X* is declared. The use of functional semantics (the single assignment rule) allows the compiler to control the order in which the values are calculated and the degree of parallelism to employ in that calculation.

3.2 Where Clauses

The provision of the *where* clause permits different operations to be used to generate the values across a domain. To preserve the single assignment rule a *where* clause would usually be placed to the right of the assignment operator and be structured so as to provide values for all elements of the variable within the domain. In DWARF the *where* clause is placed to the left of the assignment operator in order to be able to group calculations by spatial or data conditions. As a result, the compiler is required to enforce single assignment and coverage requirements. The form of *where* clause adopted in DWARF comprises a boolean expression which is evaluated at each point in the

domain of its definition and an enclosed set of calculations, which are executed everywhere the boolean is true.

3.3 Transforms

Transforms provide a method for extracting values from any domain and locating them at the nodes of another domain. The syntax involves the use of an index expression following any variable or expression. These expressions combine the index variables of the domain into which the values are returned to calculate the index values of the domain from which the values are taken. For example, consider the following domains and variables.

domain <i>world</i> (I:1..N, J:1..N, K:1..N)	with variables P and T
domain <i>surface</i> (R:1..N, S:1..N)	with variables SP and ST

The calculation

$$SP = P [R, S, 0]$$

is a domain transformation where the index variables in the *surface* domain are used to select an element from the *world* domain. The language requires that such expressions generate valid selection values for all possible R and S. The compiler therefore limits the kinds of expressions allowed in transforms to the simple linear forms

$$\langle \text{const} \rangle * \langle \text{index} \rangle + \langle \text{const} \rangle$$

The previous example is a cut through a higher dimensioned domain. A cut can be made in any way at all as in

$$\begin{aligned} SP &= P [S, 0, R] \\ \text{or} \quad SP &= P [S, S, R] \quad (\text{a diagonal cut}) \end{aligned}$$

Values can be broadcast from a smaller domain to a larger as in

$$P = SP [I, J] + \dots$$

in which all points in the domain *world* apply their I and J index values to select an element from the *surface* domain. In this case values will be duplicated in the K dimension. Broadcast can be specified within the same domain by selecting the same value for many nodes as in

$$P = P [I, J, 0]$$

Quite arbitrary permutations are possible as in

$$T = T [N+1-I, J, K];$$

in which all points in the *world* domain select another value from the same domain

3.4 Reduction Operators

To assist with the manipulation of values across a domain, DWARF supports a range of reduction operators. Reduction operators can appear in any calculation and consist of the operator followed by an expression and a list of selected index variables for the domain of the expression:

$$\langle \text{redop} \rangle (\langle \text{expr} \rangle \mid \langle \text{list of index variables} \rangle)$$

For example, given a three dimensional domain, *world*, with indices I, J and K,

$$\text{sum} (\text{Pressure} \mid K)$$

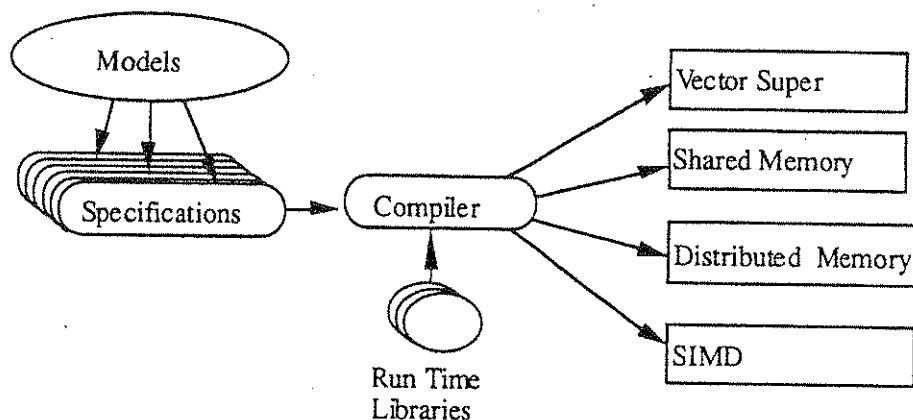
sums the values of the Pressure variable along the K dimension and then broadcasts the summed value back to all nodes in that dimension. Effectively a two dimensional set of values is calculated by summing in the third dimension and all nodes in *world* obtain a copy of the appropriate summed value. This strategy of rebroadcast means that all values exist in an explicitly declared domain. The expression

$$\text{sum} (\text{Energy} \mid I, J, K)$$

calculates a scalar value accumulating the Energy at all points in the *world* domain and broadcasts the final value back to all nodes in *world*. Reductions over parts of a domain can be implemented using *where* clauses

4. OVERALL STRUCTURE OF THE DWARF SYSTEM

The basic structure of the DWARF system is indicated in the following diagram.



The machines currently under investigation for supporting the climate models include the CRAY-YMP, as new models must be capable of demonstrating equivalent performance to FORTRAN models, the massively parallel SIMD architectures of the Connection Machine CM-2 and the Maspar MP-2, distributed memory multiprocessors such as Transputer arrays, shared memory multiprocessors such as the Silicon Graphics and more novel mesh and multibus architectures which mirror the communication pattern underlying the models.

5. A SHALLOW WATER MODEL

The code given in this section shows some of the syntax of the language by encoding the standard shallow water model. Points to note include the use of the domain variables, the definition of generalised functions, the use of *even* and *odd* selectors in variable declarations to control the mapping of variables onto partial grids as required by the finite difference scheme and the implicit embedding of calculations in the time loop. The code defines a series of time step calculations applied to the fundamental variables. The derived variables are essentially local to each time step.

The shallow water equations describe the motion of an incompressible fluid with a free surface, with the constraint that the horizontal scales of motion are much larger than the vertical. For an incompressible fluid the horizontal pressure gradient is independent of depth and will remain so if the initial velocity field is independent of depth. This restriction is clearly only reasonable for a shallow layer. Thus the shallow water equations are a one layer model of fluid motion, see Pedlosky (1979) or Haltiner (1980).

The equations may be applied on any domain from a small tank to the global atmosphere or ocean provided it is modelled as a single layer. Applied to the atmosphere the shallow water equations can be taken to represent the flow averaged through the troposphere. Though obviously a great simplification of the full range of dynamics and processes of the real atmosphere (and as simulated in modern climate and weather prediction models) they are still very useful. On the largest horizontal scales atmospheric motions are approximately barotropic (invariant with height) and the shallow water equations are a reasonable approximation to the dynamics. Even simpler models (with fixed upper and lower boundaries) were used for the earliest numerical weather prediction experiments and showed some skill (Charney 1950).

The shallow water equations are a favoured choice for experiments with various model structures and numerical schemes. Although a very simple representation of the atmosphere they do include the two types of horizontal wave motion important in more realistic models, gravity waves and Rossby waves. They are also useful for experiments in parallel computing because they include much of the structure (and so problems with data communication and synchronisation) of more complicated models.

5.1 Basic equations

The simplest form of the shallow water equations is

$$\frac{d\mathbf{V}}{dt} = -f\mathbf{k} \times \mathbf{V} - \nabla\Phi \quad (1)$$

$$\frac{d\Phi}{dt} = -\Phi \nabla \cdot \mathbf{V} \quad (2)$$

where $\mathbf{V}$ is the horizontal vector wind, Φ the geopotential height (the geometric height times the gravitational acceleration), $\mathbf{k}$ the vertical unit vector and f the Coriolis parameter (zero in a non-rotating frame). Expanding the total time derivatives gives

$$\frac{\partial \mathbf{V}}{\partial t} = -(\xi + f)\mathbf{k} \times \mathbf{V} - \nabla \left(\Phi + \frac{\mathbf{V} \cdot \mathbf{V}}{2} \right) \quad (3)$$

$$\frac{\partial \Phi}{\partial t} = -\nabla \cdot (\Phi \mathbf{V}) \quad (4)$$

where $\xi = \mathbf{k} \cdot \nabla \times \mathbf{V}$ is the vertical component of the vorticity.

Even for this very simple model the equations are nonlinear and cannot be solved directly, requiring the use of numerical methods. One approach is to discretise the equations on a grid, replacing the spatial derivatives by finite differences. The other main approach is a spectral method where the fields are defined as sums over a set of suitable basis functions. The derivatives are evaluated analytically from the basis functions, giving more accuracy, subject of course to the problems of truncation. Spectral methods are popular in meteorological modelling because the basis functions usually chosen, the spherical harmonics, are similar to the natural large scale modes of the atmosphere, and so can give an accurate representation of the flow with relatively few degrees of freedom.

Both finite difference and spectral models treat the time integration in the same way, replacing the time derivative by a finite difference form. Simple leapfrog time integration is usually used, in order to ensure numerical stability, though more complicated schemes are possible. Using $()^n$ to denote time level n , this scheme can be written as

$$X^{n+1} = X^{n-1} + 2\Delta t \left(\frac{dX}{dt} \right)^n \quad (5)$$

where $(dX/dt)^n$ is the time tendency of X at the central time, n . A simpler approach using only a single time level is not possible as such schemes are unconditionally numerically unstable for wave motion.

The size of the time step that may be used depends on the model resolution and the speed of the fastest moving waves. In meteorological models these are the gravity waves which are of less interest than the slower Rossby waves. So-called semi-implicit schemes allow a longer time step by treating the gravity waves implicitly, effectively slowing them down. This does not adversely affect the Rossby waves. Such a scheme is trivial in a spectral model of the shallow water equations but in a grid point model involves solving a Helmholtz equation. Iterative relaxation type solutions to this will have much the same structure as the model itself but alternative direct methods of solution will introduce new problems for parallelisation (similar to those in spectral models).

5.2 Grid point model

The grid point model is based on that of Sadourny (1975) and uses the shallow water equations in a slightly different form. Equation (3) is modified to

$$\frac{\partial \mathbf{V}}{\partial t} = -\eta \times (\Phi \mathbf{V}) - \nabla \left(\Phi + \frac{\mathbf{V} \cdot \mathbf{V}}{2} \right) \quad (6)$$

where $\eta = (\mathbf{k} \cdot \nabla \mathbf{V}) / \Phi$ is the potential vorticity.

This form of the equations makes it easier to design a finite difference scheme that conserves potential vorticity. This is one of the properties of the full equations which it may be desirable to preserve for reasons of accuracy and stability. The model is cast on a doubly periodic plane rather than a sphere and the Coriolis term is dropped. The grid is staggered with the velocities defined on

different points to the geopotential and vorticity. This is desirable for numerical reasons and has no effect on parallelisation.

5.3 Shallow in DWARF

The code in the appendix represents the shallow grid model as listed by a prototype DWARF compiler. This compiler is capable of the syntax analysis required by the language and code generation of that subset of the language needed by the shallow wave equation model. The prototype is an important stage in system development as it provides for the syntax and semantic verification of models expressed in DWARF. The objective behind its development is to show that the language covers the problem domain by encoding a range of models and verifying that the resulting programs are legal examples of DWARF code. In addition, the cray code generator permits a performance evaluation between models expressed in FORTRAN and models expressed in DWARF. The grid model listed executes with effectively identical flop rates as the original hand tuned FORTRAN code (an actual difference of 0.2%) demonstrating that the abstraction provided by DWARF is not made at the expense of performance.

6. SUMMARY

The current state of the project involves three concurrent activities. First, the language is undergoing a process of refinement based on the encoding of various simple models and sections of more complex models. This effort is directed to ensuring coverage of the modelling domain. Second a parser has been written to compile the models written in the language into an internal form suitable for analysis. This tests various aspects of the syntactic nature of the language and also allows the latex listing generator to be constructed and its appeal evaluated. Finally a CRAY-YMP code generator capable of analysing and compiling that part of the language needed for the shallow wave model has been constructed. The intent is to evaluate the performance characteristics of models developed in the language compared with their original FORTRAN equivalents.

As reported, the initial results of this work are extremely encouraging. Immediate progress will involve the encoding of more significant models in DWARF and the design and engineering of analysis steps capable of supporting compilation of the complete language to efficient parallel code. By the end of the year a target parallel architecture will be selected and the development of a tuned code generator commenced. The project's objectives entail the development of a climate modelling system providing tuned performance on a selected parallel architecture by the end of 1992.

ACKNOWLEDGMENTS

The work described in this paper was conducted as a collaborative project between the CSIRO Divisions of Information Technology and Atmospheric Research as part of the DIT High Performance Computation Project. The High Performance Computation Project is a joint project between the Commonwealth Scientific and Industrial Scientific Organisation (CSIRO) Division of Information Technology and the Royal Melbourne Institute of Technology (RMIT).

REFERENCES

- Chang, P.S. and Egan, G.K., (1990a) *An Implementation of a Barotropic Numerical Weather Prediction Model in the Functional Language SISAL*, Proceedings of the Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, SIGPLAN Notices, 25(3) 109-117.
- Chang, P.S. and Egan, G.K., (1990b) *Parallelising an Australian Region NWP Model*, Proc. 3rd. Australian Supercomputer Conference, December 1990.
- Pedlosky, J., (1979), *Geophysical Fluid Dynamics*, Springer-Verlag.
- Haltiner, G.J. and Williams, R.T. (1980), *Numerical Prediction and Dynamic Meteorology*, Wiley.
- Charney, J., Fjortoft, R. and von Neumann, J., (1950), *Numerical integration of the barotropic vorticity equation*, Tellus, 2, 237-254.
- Sadourny, R., (1975), *The dynamics of finite-difference models of the shallow water equations*, J. Atmos. Sci., 32, 680-689.

APPENDIX

MODEL : shallow

Constants

Δ_x	1e+06	Real	m
Δ_y	1e+06	Real	m
Δ_t	90	Real	s
alpha	0.001	Real	
end_run	86400	Real	
size	64	Integer	

Domains

world [I : 1..size [ew], J : 1..size [ns]] *Torus*

Functions

$\delta_x (F \text{ world})$	$\Rightarrow$ world	$: (F.e - F.w) / \Delta_x$
$\delta_y (F \text{ world})$	$\Rightarrow$ world	$: (F.n - F.s) / \Delta_y$
$\overline{F \text{ world}}^x$	$\Rightarrow$ world	$: (F.e + F.w) * 0.5$
$\overline{F \text{ world}}^y$	$\Rightarrow$ world	$: (F.n + F.s) * 0.5$

Fundamental Variables

P	world [Even, Even]	Real	$m^2 s^{-2}$
Pstar	world [Even, Even]	Real	$m^2 s^{-2}$
Ptend	world [Even, Even]	Real	$m^2 s^{-3}$
u	world [Odd, Even]	Real	ms
ustar	world [Odd, Even]	Real	ms
utend	world [Odd, Even]	Real	ms^{-2}
v	world [Even, Odd]	Real	ms
vstar	world [Even, Odd]	Real	ms
vtend	world [Even, Odd]	Real	ms^{-2}
time		Real	s

Derived Variables

H	world [Even, Even]	Real	$m^2 s^{-2}$
U	world [Odd, Even]	Real	$m^3 s^{-3}$
V	world [Even, Odd]	Real	$m^3 s^{-3}$
Z	world [Odd, Odd]	Real	sm^{-2}
Kenergy		Real	$m^4 s^{-4}$
Penstrophy		Real	m^{-2}
AXYP	world [Odd, Odd]	Real	sm^{-2}

APPENDIX

Calculations

$$\begin{aligned}
 U &= \bar{P}^x * u \\
 V &= \bar{P}^y * v \\
 H &= P + 0.5 * (\bar{u} * \bar{u}^x + \bar{v} * \bar{v}^y) \\
 AXYP &= \bar{P}^y^x \\
 Z &= (\delta_x(v) - \delta_y(u)) / AXYP \\
 utend &= \bar{v}^y^x * \bar{Z}^y - \delta_x(H) \\
 vtend &= (-\bar{u}^y^x) * \bar{Z}^x - \delta_y(H) \\
 Ptend &= -\delta_x(U) - \delta_y(V) \\
 Kenergy &= \sum_{I,J} (P * H) \\
 Penstrophy &= \sum_{I,J} (Z * Z * AXYP)
 \end{aligned}$$

Initially

time = 0
 Input(P, u, v)

First Step

new time = Δ_t
 new P = $P + \Delta_t * Ptend$
 new u = $u + \Delta_t * utend$
 new v = $v + \Delta_t * vtend$
 Pstar = P
 ustar = u
 vstar = v
 Output(Kenergy, Penstrophy)

While time < end_run do

new time = time + Δ_t
 new P = $oldPstar + 2 * \Delta_t * Ptend$
 new u = $oldustar + 2 * \Delta_t * utend$
 new v = $oldvstar + 2 * \Delta_t * vtend$
 Pstar = $P + \alpha * (newP - 2 * P + oldPstar)$
 ustar = $u + \alpha * (newu - 2 * u + oldustar)$
 vstar = $v + \alpha * (newv - 2 * v + oldvstar)$
 Output(Kenergy, Penstrophy)

Finally

Output(Kenergy, Penstrophy, P, u, v)