JOINT ROYAL MELBOURNE INSTITUTE OF TECHNOLOGY AND
COMMONWEALTH SCIENTIFIC AND INDUSTRIAL RESEARCH ORGANISATION
PARALLEL SYSTEMS ARCHITECTURE PROJECT

An Implementation of a Barotropic Numerical Weather Prediction Model in the Functional Language SISAL

TR 118 088 R

Pau S. Chang
Gregory K. Egan

Department of Communication and Electrical Engineering
Royal Melbourne Institute of Technology
124 Latrobe St
Melbourne 3000

Version 1.0 Original Document 1/8/89

ABSTRACT

Numercial Weather Prediction (NWP) is acknowledged as being of vital importance to Australian and world economies. The demand that NWP places on computing system performance has increased dramatically since the introduction of computer systems and is still growing. The paper describes the parallel implementation of a one-level barotropic spectral NWP model using the high level functional dataflow language SISAL. The implementation technique is discussed and results for an Encore Multimax multiprocessor, and a Sun 3/260 are presented.

Introduction

Numercial Weather Prediction (NWP) is acknowledged as being of vital importance to Australian and world economies. The demand that NWP places on computing system performance has increased dramatically since the introduction of computer systems and is still growing. As technological limits are approached in component performance, many computer manufacturers are turning to multiprocessor configurations to obtain increased performance. Although the underlying application may exhibit large amounts of inherent parallelism, in many cases this has been lost in formulation for sequential and vector systems. Therefore, the computational techniques of weather modeling will have to be revised to take advantage of the technology of parallel processing. Additionally there is a strong suggestion that existing language systems will prove inadequate for the new multiprocessors. Hence, having developed new more power techniques, a parallel computing language is required to implement these techniques in order to effectively express the inherent parallelism of the weather model. The purpose of this paper subsequently is to describe our implementation of a one-level barotropic spectral NWP model [1] using the high level parallel functional dataflow language SISAL [2].

In order to investigate the feasibility of the implementation in SISAL, a one-level model is adopted instead of the very large multi-level model. The implementations using a direct transliteration approach from FORTRAN to SISAL, and a new parallel approach are described and results for an ENCORE Multimax multiprocessor, and a SUN 3/260 are presented. The initial results, obtained using an Optimising SISAL Compiler [3], are referenced to the baseline runtime of the original FORTRAN (sequential) formulation. The analysis in terms of parallelism profiles, speedup curves and model size, and their impact on achieved performance lead to a continuing study of issues related to the effective use of current and next generation multiprocessors [4].

1. Weather Prediction Model

In contrast to the usual "grid points" models which represent parameters as grid points in space, the spectral model studied here represents these parameters in terms of spatial basis functions. The model size is expressed in terms of the *number of resolution* (or *wave number truncation*), *J*, and the associated *number of Gaussian latitudes*, *ilat*, and *number of longitudinal points*, *ilong*, on each latitude. Apart from the interesting technical features of the model, access to a FORTRAN implementation and its associated data sets was an important factor in the choice of the model for this study. A full description of the model setting out its advantages, mathematical algorithm and presenting the results which were obtained on an IBM 360/65, is given by Bourke in [1].

1.1 Mathematical Description of the Model

For the purpose of this feasibility study, the one-level spectral model is an approximate representative of a full multi-level spectral model. In the latter, a lot of sequential code will probably be introduced when the treatment of more complicated physics is considered. Nevertheless, inspection of the equations decribing the one-level model suggests very high potential concurrency. In its primitive form, the model is expressed in terms of the vorticity and divergence of the horizontal wind field as shown in Figure 1.

Integration of the primitive equations is facilitated by a spectral grid transform technique which arises in evaluation of the nonlinear products $U\nabla^2\psi$, $V\nabla^2\psi$, $U\Phi'$ and $V\Phi'$ in equations 4, 5 and 6. Briefly, the first step of this technique is to obtain the truncated expansions for approximating the stream function, geopotential height and two derived wind fields U and V. This is followed by a fast Fourier transformation of these fields to the Gaussian latitude-longitude grid on the globe, and direct multiplications in the grid domain to obtain the nonlinear products. An inverse fast Fourier transformation of these terms is then performed, followed by another transformation back to the spectral domain.

Definitions of symbols used:

$\underline{V}$	=	wind vector (east U and north V)	ψ	=	stream function
∇	=	horizontal gradient operator	χ	=	velocity potential
$\underline{k}$	=	vertical unit vector	Φ'	=	time dependent perturbation field
ζ	=	vertical component of relative vorticity	$\overline{\Phi}$	=	geopotential height of the surface
D	=	horizontal divergence	$\overline{\Phi}$	=	global mean geopotential
J	=	wave number truncation (resolution)			
Ω	=	angular velocity of earth			
a	=	radius of earth			

ϕ and λ are spherical coordinates

$$\zeta = \underline{k} \cdot \nabla \times \underline{V} = \nabla^2\psi \quad (1)$$

$$D = \nabla \cdot \underline{V} = \nabla^2\chi \quad (2)$$

$$\Phi = \overline{\Phi} + \Phi' \quad (3)$$

$$\frac{\delta(\nabla^2\psi)}{\delta t} = \frac{-1}{a\cos^2\phi}\left[\frac{\delta(U\nabla^2\psi)}{\delta\lambda} + \cos\phi\frac{\delta(V\nabla^2\psi)}{\delta\phi}\right] - 2\Omega\left(\sin\phi\,\nabla^2\chi + \frac{V}{a}\right) \quad (4)$$

$$\frac{\delta(\nabla^2\chi)}{\delta t} = \frac{1}{a\cos^2\phi}\left[\frac{\delta(V\nabla^2\psi)}{\delta\lambda} - \cos\phi\frac{\delta(U\nabla^2\psi)}{\delta\phi}\right] + 2\Omega\left(\sin\phi\,\nabla^2\chi - \frac{U}{a}\right) - \nabla^2\left[\frac{U^2+V^2}{2\cos^2\phi} + \Phi'\right] \quad (5)$$

$$\frac{\delta\Phi'}{\delta t} = \frac{-1}{a\cos^2\phi}\left[\frac{\delta U\Phi'}{\delta\lambda} + \cos\phi\frac{\delta V\Phi'}{\delta\phi}\right] - \overline{\Phi}\,D \quad (6)$$

$$U = \frac{-\cos\phi}{a}\frac{\delta\psi}{\delta\phi} + \frac{1}{a}\frac{\delta\chi}{\delta\lambda} \quad (7)$$

$$V = \frac{1}{a}\frac{\delta\psi}{\delta\lambda} + \frac{\cos\phi}{a}\frac{\delta\chi}{\delta\phi} \quad (8)$$

Figure 1: Primitive form of the mathematical model.

2 Direct Transliteration Approach

Our initial approach was to perform a direct transliteration of the FORTRAN codes into SISAL. This results in a sequential SISAL implementation because the FORTRAN implementation was sequentially conceived.

2.1 Sequential Implementation

The original FORTRAN implementation of the model was provided by the Department of Meteorology at the University of Melbourne. The flow chart for the FORTRAN implementation, thus the sequential SISAL implementation, is given in Figure 2.

The model consists of an *initialisation* section and a *timeloop* section. In the initialisation, all lookup tables, indexing arrays and the initial values of four spectral fields of interest are computed and built whereas the future states of the fields are computed in the timeloop section.

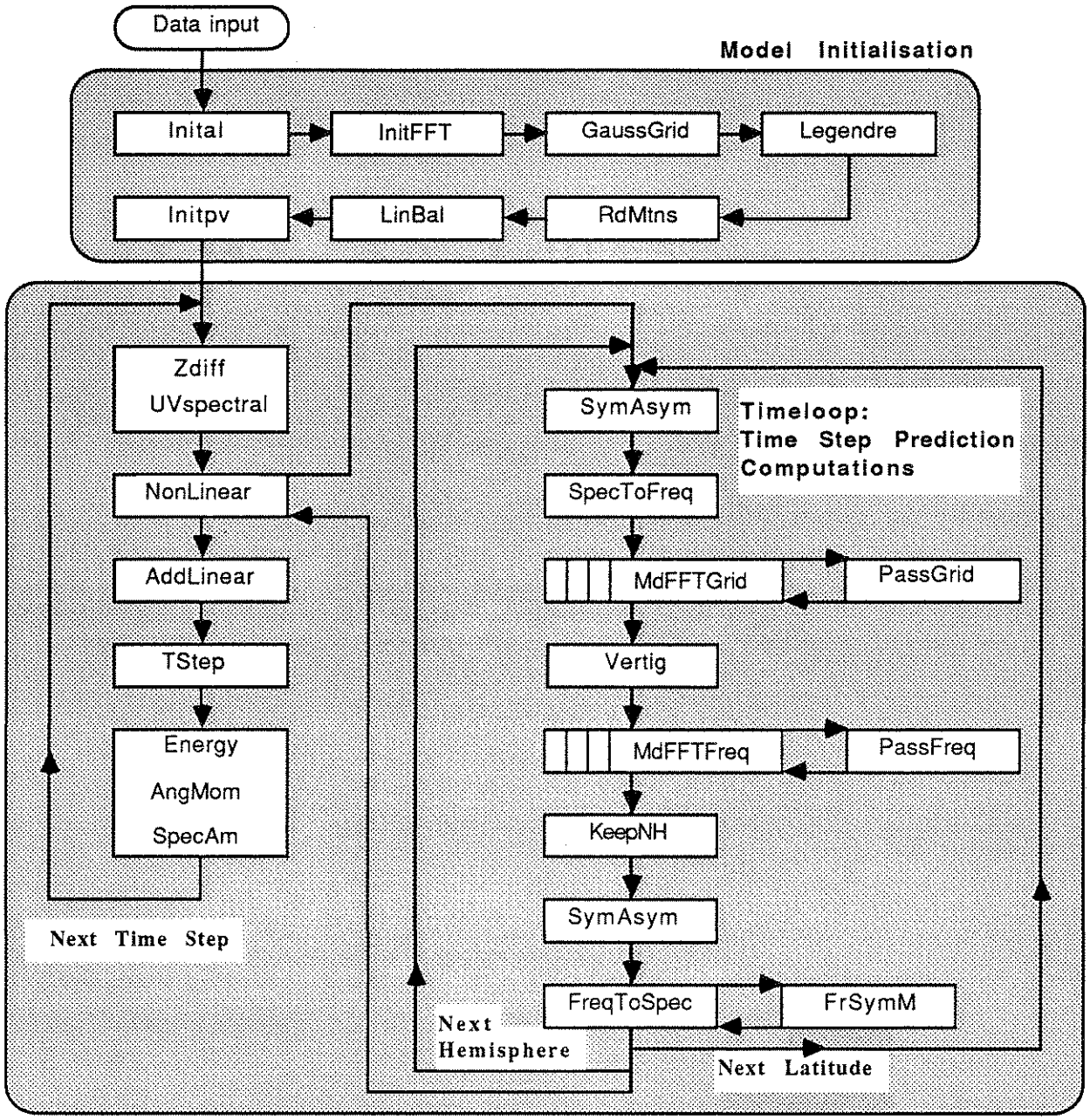

Figure 2: Flowchart for the FORTRAN and sequential SISAL implementations

The function of the major subroutines in the sequential implementations are detailed below.

Inital initialise essential model variables and generate indexing arrays of orthogonal spherical harmonics.

InitFFT generate tables for the sines and cosines required by FFTs.

GaussGrid generate the cosine of the colatitudes and the weights for the Gaussian quadrature.

Legendre compute the spherical harmonics for each latitude from the associated Legendre polynomial of the first kind:

$$\int_{-\pi/2}^{\pi/2} P_r^m(\sin\phi)\, P_r^m(\sin\phi)\, \cos\phi\, d\phi = 1$$

where P_r^m $(\sin\phi)$ = Normalised Legendre Polynomial

RdMtns read the topography (mountains) of the globe in spectral form imposing linear balance condition (dD/dt).

LinBal compute the starting geopotential field.

Initpv compute the starting tendencies of the spectral stream function, divergence and geopotential fields; the spectral forms of the fields are complex expansion coeffecients.

This completes the initialisation section of the model. The loop body of the other section, the timeloop, is comprised of the following subroutines:

Zdiff compute the spectral time dependent geopotential perturbation field

UVspectral compute the two spectral wind fields.

NonLinear compute the transformations of fields latitude by latitude and, for each latitude, hemisphere by hemisphere.

- **SymAsym** if computing for southern hemisphere, compute the anti-symmetrical spherical harmonics from the symmetrical spherical harmonics for northern hemisphere.
- **SpecToFreq** compute truncated expansions of the fields.
- **MdFFTGrid** compute FFT of each truncated field.
- **Vertig** compute nonlinear products by direct multiplication.
- **MdFFTFreq** inverse FFT of these products.
- **KeepNH** retain the northern hemisphere fields until the southern hemisphere fields have been computed.
- **FreqToSpec** accumulate intermediate non linear terms for each latitude in spectral form.

AddLinear add the linear and nonlinear terms.

TStep perform a model timestep in the spectral model.

Energy check energy conservation.

AngMom check angular momentum conservation.

Specam check conservation of vorticity, divergence and height.

For a model size of J = 30 for example, each time step is 30 minutes; hence 48 iterations of the timeloop are needed to perform a 24-hour forecast. This section, as a result, dominates the computation time of the model.

The parallelism profile has indicated that the most computationally intensive routines are inside **Nonlinear** where various transformations are performed. The form of computation of this critical region for each time step is a multi-nested sequential **For** loops as illustrated in Figure 3.

```
For each hemisphere
    For each latitude
        evaluate the appropriate signs of the spherical harmonics
                :
        For each .....
        For each .....
        For each longitude point or element in a spectral field
            compute the new value of the point or element
        For each .....
        For each .....
                :
        Next point or element
    Next latitude
Next hemisphere
```

*Figure 3: Multi-nested sequential **For** loops in **Nonlinear***

2.2 Mapping from FORTRAN to SISAL

A typical mapping from FORTRAN to SISAL for **SpecToFreq**, one of the few subroutines which constitutes the innermost **For** loops in **Nonlinear**, is given in Figure 4. This subroutine evaluates a transformation to compute the truncated expansions of four fields of interest. The mappings resulted in two inner parallel For loops, which performed summations, inside an outer sequential loop which updated four arrays. With many other codes like **SpecToFreq**, some worse, residing inside another two sequential **For** loops (***For** each hemisphere* and ***For** each latitude*) as shown in Figure 3, the result was a large number of complicated sequential array updates, thus array copyings, in multi-nested sequential loops. Hence the memory capacity required for this implementation was many times larger than that for the FORTRAN implementation.

```
      subroutine SpecToFreq
      do 20 m = 1, mx
      mi = m + m
      mr = mi - 1
      pg(mr) = 0.0
      pg(mi) = 0.0
      zg(mr) = 0.0
      zg(mi) = 0.0
      do 30 j = jx, 1, -1
      if (j .eq. 1 .and. m .eq. 1) go to 30
      jm = kmjx(m) + j
      jmi = jm + jm
      jmr =jmi - 1
      jmx =kmjxx(m) + j
      pg(mr) = pg(mr) + alp(jmx) * pri(jmr)
      pg(mi) = pg(mi) + alp(jmx) * pri(jmi)
      zg(mr) = zg(mr) + alp(jmx) * zri(jmr)
      zg(mi) = zg(mi) + alp(jmx) * zri(jmi)
30    continue
20    continue
c
      do 120 m = 1, mx
      mi = m + m
      mr = mi - 1
      ug(mr) = 0.0
      ug(mi) = 0.0
      vg(mr) = 0.0
      vg(mi) = 0.0
      do 130 j = jxx, 1, -1
      jmx= kmjxx(m) + j
      jmi = jmx + jmx
      jmr = jmi - 1
      ug(mr )= ug(mr) + alp(jmx) * uri(jmr)
      ug(mi) = ug(mi) + alp(jmx) * uri(jmi)
```

```
            vg(mr) = vg(mr) + alp(jmx) * vri(jmr)
            vg(mi) = vg(mi) + alp(jmx) * vri(jmi)
      130   continue
      120   continue
            return
            end
```

(a) *FORTRAN implementation of* ***SpecToFreq***

```
FUNCTION SpecToFreq   (jx, mx, jxx : integer; kmjx, kmjxx : ArrInt1;
                       alp : ArrReal1; pri, zri, uri, vri : ArrReal1
                       RETURNS ArrReal1, ArrReal1, ArrReal1, ArrReal1)
LET
pg, zg, ug, vg :=
    FOR INITIAL
    m := 1;
    pg := ARRAY_fill(1, mx * 2, 0.0);
    zg := ARRAY_fill(1, mx * 2, 0.0);
    ug := ARRAY_fill(1, mx * 2, 0.0);
    vg := ARRAY_fill(1, mx * 2, 0.0);
    WHILE m <= mx REPEAT
    m := old m + 1;
    mi := old m * 2;
    mr := mi - 1;
    pgmr, pgmi, zgmr, zgmi :=
      FOR j IN 1, jx
      jm := kmjx[old m] + j;
      jmi := jm * 2;
      jmr := jmi - 1;
      jmx := kmjxx[old m] + j;
      RETURNS VALUE of SUM IF old m = 1 & j = 1 THEN 0.0
                           ELSE alp[jmx] * pri[jmr]
                           END IF
              VALUE of SUM IF old m = 1 & j = 1 THEN 0.0
                           ELSE alp[jmx] * pri[jmi]
                           END IF
                           VALUE of SUM IF old m = 1 & j = 1 THEN 0.0
                           ELSE alp[jmx] * zri[jmr]
                           END IF
              VALUE of SUM IF old m = 1 & j = 1 THEN 0.0
                           ELSE alp[jmx] * zri[jmi]
                           END IF
      END FOR;
    pg := old pg[mi :pgmi; mr:pgmr];
    zg := old zg[mi :zgmi; mr:zgmr];
    ugmr, ugmi, vgmr, vgmi :=
      FOR j IN 1, jxx
      jmx := kmjxx[old m] + j;
      jmi := jmx * 2;
      jmr := jmi - 1;
      RETURNS VALUE of SUM alp[jmx] * uri[jmr]
              VALUE of SUM alp[jmx] * uri[jmi]
              VALUE of SUM alp[jmx] * vri[jmr]
              VALUE of SUM alp[jmx] * vri[jmi]
      END FOR;
    ug := old ug[mi : ugmi; mr : ugmr];
    vg := old vg[mi : vgmi; mr : vgmr];
    RETURNS VALUE of pg
            VALUE of zg
            VALUE of ug
            VALUE of vg
    END FOR;
IN pg, zg, ug, vg
END LET
END FUNCTION
```

(b) *Sequential SISAL implementation*

Figure 4: Direct transliteration of the FORTRAN implementation of SpecToFreq to SISAL

While many FORTRAN control structures mapped readily into SISAL, as was anticipated, some difficulties were encountered with **common** and **equivalence** statements and the implicit mappings from **real** to **complex** number representations using these statements because SISAL does not provide global structures or implicit mappings. Also with the same **common** and **equivalence** statements installed in the subroutines, the resulting *side effects* throughout the FORTRAN program made it difficult for the context of the model to be understood. In addition, SISAL does not currently provide an intrinsic complex number type although this may be added in future revisions.

The adoption of a direct transliteration of the original code and the need to express operations on complex numbers explicitly resulted in a SISAL formulation which was long (approximately 3100 lines) compared to the original FORTRAN. The links between the mathematical formulation and both the FORTRAN and the directly transliterated SISAL codes were also obscure.

2.3 Results for the Direct Transliteration Approach

The SISAL and FORTRAN programs were run on a Sun 3/260 (68020/68881) and an Encore Multimax with APC (32332/32081) processors. The results were obtained using the standard f77 FORTRAN compiler released with the Sun 3/260 (SunOS 3.5) and Encore Multimax (Umax 4.2, rel 3.3.0), and the Optimising SISAL Compiler (OSC) [3] from the Colorado State University. The FORTRAN compilations were performed with inlining of floating point functions (Sun 3/260) and optimisation.

The model size chosen was a realistic resolution with J = 30 [1]. The run times of the model with one iteration of the timeloop at this resolution on the Sun 3/260 and the Encore Multimax using one processor are tabulated in Table 1. The runtime for multiple processors are plotted in Figure 5b whereas the concurrency profile is shown in Figure 5a. The results indicate that this approach is very inefficient because the code iteratively progressed through the transformation section latitude by latitude, and hemisphere by hemisphere in each latitude, before obtaining the final nonlinear terms of the new spectral fields. This resulted in the loss of parallelism due to excessive copying and sequential updating of arrays (36% of computation time). Consequently, the implementation technique has to be reformulated.

	Time (Seconds)	
	f77 -O	OSC
Sun 3/260	94.5	272.1
Encore Multimax	71.0	773.1

Table 1: Single processor run times for FORTRAN and sequential SISAL.

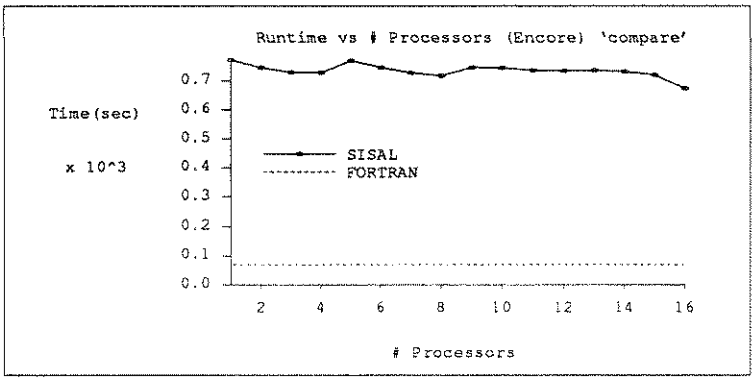

Figure 5a: Multiple processor run times for FORTRAN and sequential SISAL

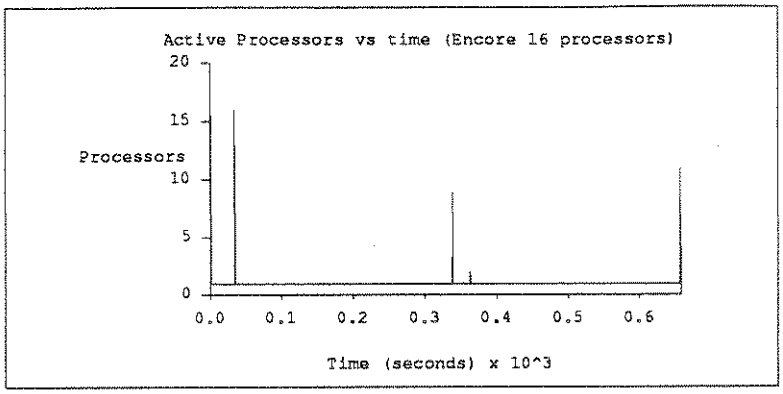

Figure 5b: Parallellism profile of the directly transliterated SISAL version

3 Parallel Implementation of the Model

Our analysis indicated that the new state for a particular field of interest in a particular time frame was dependent on its values in the previous time frame, but the values of the state were themselves independent of each other. By examination of the mathematical model consequently, it could be seen that a parallel formulation of **Nonlinear**, which could be viewed as a whole "slice" of the globe, may be implemented:

FOR both hemispheres **and all** latitudes
  FOR all longitude points *or elements in a spectral field*
    compute the new values of the field

This results in a new implementation which exposed the inherent parallelism of the model. For example, the loops

FOR both hemispheres **and all** latitudes
  SpecToFreq

may be absorbed into subroutine **SpecToFreqSphere**. Furthermore, **SpecToFreq** itself has been rewritten so that its outer loop also generates successively the individual indices of the four arrays, thus making possible the use of a parallel loop construct, in addition to the two parallel inner summations. Figure 6 shows how this technique have been implemented where entirely new arrays were created directly using multi-nested parallel **FOR** loops instead of old arrays being updated in multi-nested sequential loops. This not only produced more concise code, the memory required for each parallel **FOR** loop in such implementation was also lower relative to that of the sequential implementation. The resulting block diagram for this formulation is presented in Figure 7.

```
FUNCTION SpecToFreqSphere   (jx, mx, jxx, ilath, ixh : integer; kmjx, kmjxx : ArrInt1;
                             alp : ArrReal3; pri, zri, uri, vri : ArrReal1
                             RETURNS ArrReal3, ArrReal3, ArrReal3, ArrReal3)
LET
pg, zg, ug, vg :=
    FOR hemi IN 1, 2 CROSS latlev IN 1, ilat / 2
     pg, zg, ug, vg:=                                        % Herein is SpecToFreq
        FOR mrmi IN 1, mx * 2
        m := (mrmi + 1) / 2;
        pg, zg :=
              FOR j IN 1, jx
              jm := kmjx[m] + j;
              jmx := kmjxx[m] + j;
              jmrjmi := jm * 2 - MOD(mrmi, 2);
              pg, zg := IF m = 1 & j = 1 THEN 0.0, 0.0
                        ELSE alp[hemi, latlev, jmx] * pri[jmrjmi], alp[hemi, latlev, jmx] * zri[jmrjmi]
                        END IF
              RETURNS  VALUE of SUM pg
                       VALUE of SUM zg
              END FOR;
```

```
            ug, vg :=
                  FOR j IN 1, jxx
                  jmx := kmjxx[m] + j;
                  jmrjmi := jmx * 2 - MOD(mrmi, 2)
                  RETURNS   VALUE of SUM alp[hemi, latlev, jmx] * uri[jmrjmi]
                            VALUE of SUM alp[hemi, latlev, jmx] * vri[jmrjmi]
                  END FOR;
            RETURNS  ARRAY of pg
                     ARRAY of zg
                     ARRAY of ug
                     ARRAY of vg
            END FOR;
      RETURNS  ARRAY of pg
               ARRAY of zg
               ARRAY of ug
               ARRAY of vg
      END FOR;
IN pg, zg, ug, vg
END LET
END FUNCTION
```

Figure 6: Parallel implementation of the truncated expansion computation

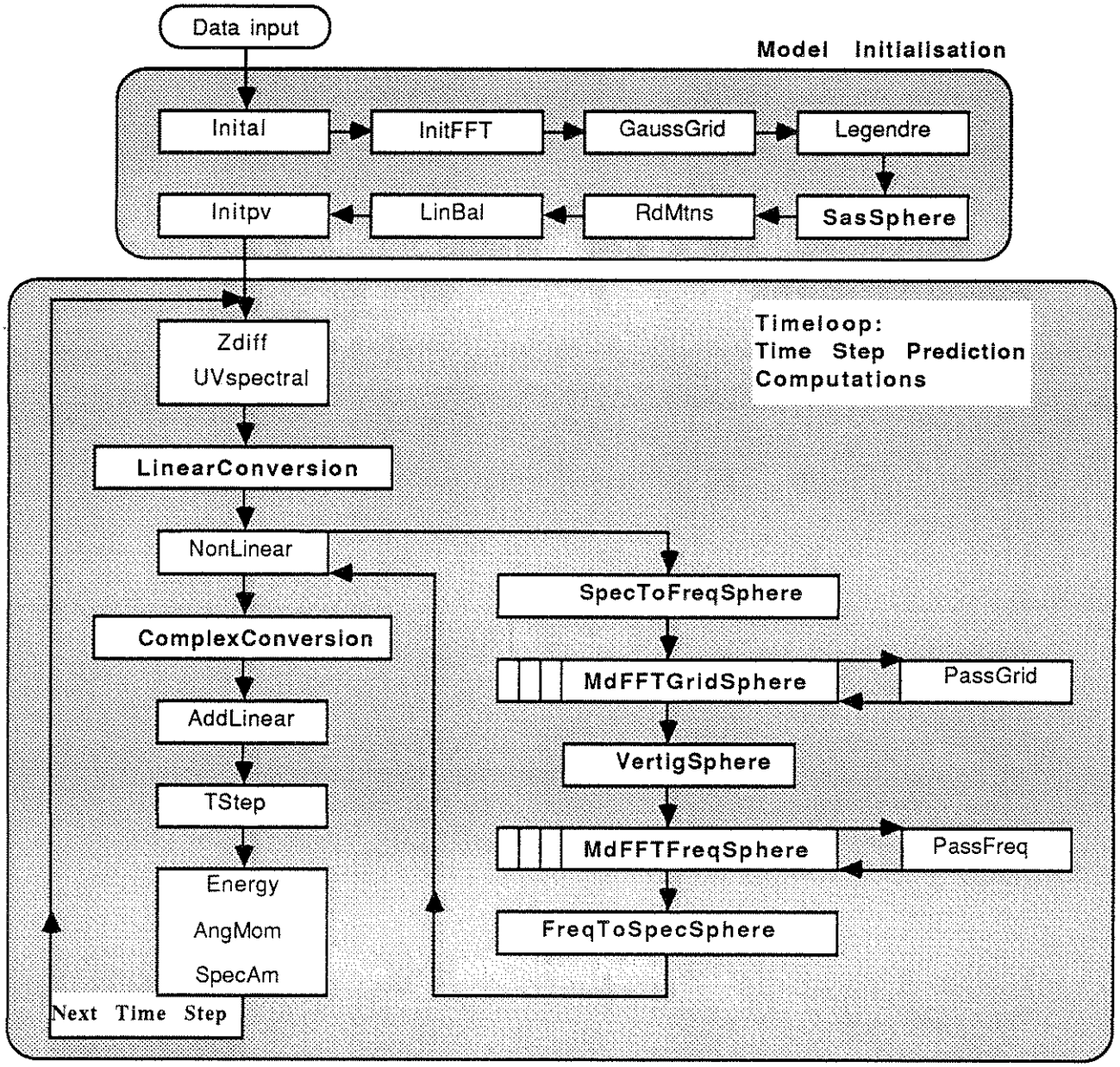

Figure 7: Flowchart for parallel SISAL implementation

In the new implementation, evaluations of both the global symmetrical and anti-symmetrical spherical harmonics have been relocated so that they are precomputed concurrently only once in **SasSphere** in the model initialisation stage. They are stored as templates and are retrieved as required in the timeloop. The algorithm in each of the subroutines **SpecToFreqSphere**, **MdFFTGridSphere**, **VertigSphere**, **MdFFTFreqSphere** and **FreqToSpecSphere** within **Nonlinear** has been rearranged as computation of the sphere in a whole "slice", and implemented using the SISAL multi-nested parallel **FOR** loop construct. All other subroutines in the timeloop also have their algorithms rearranged so that new arrays are created intead of old ones updated. The resulting new codes are shorter and show a direct link to the corresponding mathematical equations.

4.1 Results for the Parallel Implementation

The model size used for the parallel implementation is the same as that for the direct transliteration approach. The run times for one iteration of the main loop at this resolution on the Sun 3/260 and the Encore Multimax using one processor are tabulated in Table 2. The runtime for multiple processors are plotted in Figure 8a. The runtime on a single Encore processor is now 106.7 seconds which indicates that SISAL's parallel implementation executes 7 times faster than its sequential implementation. Additionally, Figure 8b indicates that scalable speedups with the number of processors used have been obtained. With 16 processors sharing the workload, the run time has been reduced to 45.3 seconds for this model size. The concurrency profile also shows a large sequential code section at the initialisation which needs to be parallelised, and a highly parallel code section which indicates that the parallelisation of the timeloop has been successful.

Nevertheless, the overall results confirms the feasibility of a parallel implementation of the adopted weather model. It is possible that a similar reformulation of the model in FORTRAN would yield some improvement.

	Time (Seconds)	
	f77 -O	OSC
Sun 3/260	94.5	74.5
Encore Multimax	71.0	106.7

Table 2: Single processor run times for FORTRAN and parallel SISAL

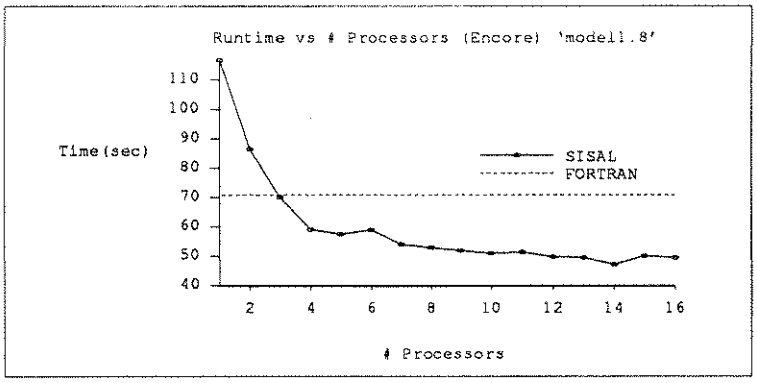

Figure 8a: Execution time profile of the new implementation

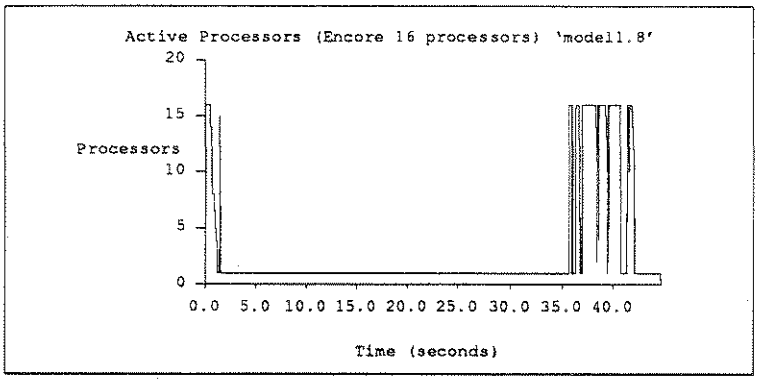

Figure 8b: Concurrency profile for the new implementation

4.2 Scalability

It is important to focus on the performance of the timeloop since this is the dominating section of the model in terms of computation time. The model sizes we adopted for benchmarking were relatively small for very accurate weather modeling. Figure 9 shows the speedup curves for various model sizes, ie J = 6 (low resolution), 15, 24 and 30 (high resolution), plotted wih the linear and 50% of linear (corresponding to 50% system utilisation) lines. It can be seen that as the model size increases so does the slope of the tail (at 16 processors) of the speedup curve. The ratio of sequential to parallel regions in the timeloop computation is reduced with increasing model size. The concurrency profiles for low resolution and high resolution models shown in Figures 10a and 10b indicate that the relative time spent in sequential regions is reduced as excess available tasks in the SISAL runtime task queue are transferred forward in time, thus extending the parallel regions. The sequential regions in the profile are currently being investigated and may due primarily to the strict implementation of arrays in SISAL. It is expected that better speedup will be obtained as the sequential code sections on the concurrency profile in Figure 10b are also parallelised.

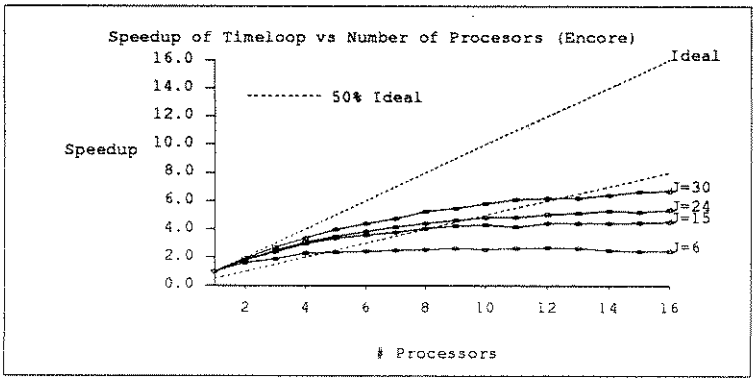

Figure 9: Speedup curves (timeloop) for various model sizes

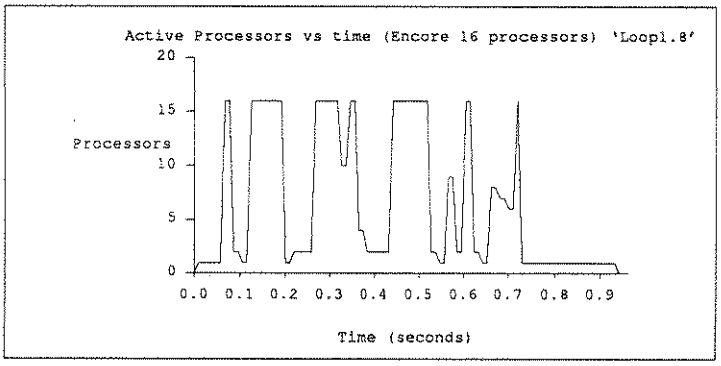

Figure 10a: Concurrency profile for J = 6

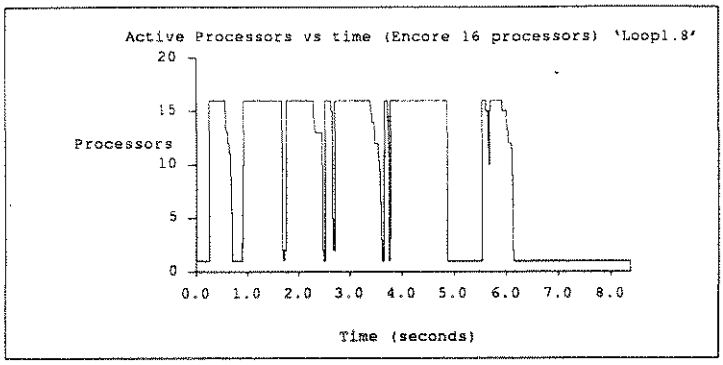

Figure 10b: Concurrency profile for J = 30

5. Conclusions

The initial results of the study described in this paper are encouraging with the SISAL single processor performance being gratifyingly close to that of FORTRAN on the Sun and Encore systems. It is anticipated that further work by ourselves and the SISAL developers will result in continued improvement in performance. We anticipate that the dataflow work associated with this study will permit exploitation of expression level concurrency in sequential regions of applications while providing the ability to throttle excessive concurrency in parallel regions [4][5]. Due to the implicit expression of concurrency with SISAL, at no time were we concerned with the underlying machine organisation.

Acknowledgements

The authors thank Drs Ian Smith and Ian Simmonds of the Department of Meteorology at the University of Melbourne for providing access to, and interpretation of, the original FORTRAN implementation of the NWP Model. They also thank Warwick Heath for his work on instrumentation, and all members of the Project team for their contributions to the work presented in this paper. The RMIT/CSIRO Parallel Systems Architecture Project is jointly funded by the Commonwealth Scientific and Industrial Scientific Organisation (CSIRO) and the Royal Melbourne Institute of Technology.

References

[1] W. Bourke, "An Efficient, One-Level, Primitive Equation Spectral Model", Monthly Weather Review, Vol. 100, No. 9, pp 683-689, Sept. 1972.

[2] McGraw et al, "SISAL: Streams and Iteration in a Single Assignment Language, Language Reference Manual", Lawrence Livermore National Laboratories, M146.

[3] D.C. Cann, "High Performance Parallel Applicative Computation", Technical Report CS-89-104, Colorado State University, Feb.1989.

[4] D. Abramson and Egan G.K, "Design Considerations for a High Performance Dataflow Multiprocessor", ACM Computer Architecture Symposium Workshop, Eilat, 1989. Prentice-Hall, in print.

[5] G.K. Egan, "Some Shallow Experiences: The Shallow Water Numerical Weather Prediction Program" Technical Report TR 118 086 R, Department of Communication And Electrical Engineering, Royal Melbourne Institute of Technology, Aug. 1989.