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

Pau S. Chang *
Gregory K. Egan +

Joint Royal Melbourne Institute of Technology and
Commonwealth Scientific and Industrial Research Organisation
Parallel Systems Architecture Project

ABSTRACT

Numerical Weather Prediction (NWP) is acknowledged as being of vital importance to economy. The demand that NWP places on computing system performance has increased dramatically since the introduction of computer systems and is still growing. This paper describes the implementation of a one-level barotropic spectral NWP model using the high level functional language SISAL. Our initial approach was a direct transliteration from FORTRAN to SISAL, the result of which was a large amount of array copying and loss of parallelism. The algorithm of the model was then rearranged to enable effective exploitation of parallelism by an optimising SISAL compiler. The parallel implementation technique and its potential for modeling with larger model sizes are discussed, and the improved results for an Encore Multimax multiprocessor are presented.

Introduction

Numerical Weather Prediction (NWP) is acknowledged as being of vital importance to economy. 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. There is a strong suggestion that existing language systems will prove inadequate for the new multiprocessors.

* Researcher, Royal Melbourne Institute of Technology, Dept of Communication and Electrical Engineering, 124 La Trobe St, Melbourne 3000, Australia. ...!uunet!munnari!koel.co.rmit.oz!rcocp

+ Professor of Computer Systems Engineering, Swinburne Institute of Technology, School of Electrical Engineering, John Street, Hawthorn 3122, Australia. ...!uunet!munnari!minyos.rmit.oz!rcoge

The purpose of this paper is to describe our implementation of a one-level barotropic spectral NWP model [2] using the high level parallel functional language SISAL (*Streams and Iteration in a Single Assignment Language*) [7]. In order to investigate the feasibility of an implementation in SISAL, a one-level model was adopted as being representative of the much larger multi-level model. Our initial implementation using a direct transliteration approach from FORTRAN to SISAL which results in a large amount of array copying and loss of parallelism, and a new parallel implementation which effectively exploits the parallelism of the model on an Encore Multimax multiprocessor are described. The results obtained using an optimising SISAL compiler, OSC [3], are referenced to the baseline run time of the original FORTRAN (sequential) formulation. The analysis in terms of parallelism profiles, speedup curves and model size, and the impact of the analysis on achieved performance lead to a continuing study of issues related to the effective use of current and next generation multiprocessors [1][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 spherical harmonics [8]. The model size is expressed in terms of the *spectral resolution number* 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 algorithms and presenting the results which were obtained on an IBM 360/65, is given by Bourke in [2].

1.1 Mathematical Description of the Model

For the purpose of this feasibility study, a one-level spectral model is representative of a full multi-level spectral model. More complicated physics is added in the latter. Nevertheless, inspection of the equations describing 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 equations 1 to 8 in Figure 1. Integration of the primitive equations is facilitated by a spectral grid transform technique which arises in the evaluation of the nonlinear products $U^2\psi$, $V^2\psi$, $U\Phi'$ 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 four 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 = geopotential height of the surface
Φ' = time dependent perturbation field
Φ^* = global mean geopotential
ζ = vertical component of relative vorticity
D = horizontal divergence
J = wave number truncation (resolution)
Ω = angular velocity of earth
a = radius of earth
ϕ and λ are spherical coordinates

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

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

$$\Phi = \Phi^* + \Phi' \quad (3)$$

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

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

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

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

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

Figure 1: Primitive form of the mathematical model [2]

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 implementation in FORTRAN 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 FORTRAN and sequential SISAL implementations of the weather model is illustrated as a flow chart in Figure 3. 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, pg, zg, ug and vg, are computed and built whereas the future states of the fields are derived in the timeloop section.

The parallelism profile has indicated that the most computationally intensive routines are inside the function **Nonlinear** where various transformations of the fields are performed hemisphere by hemisphere, and for each hemisphere, latitude by latitude. The form of computation of this critical region for each time step is deeply nested sequential **For** loops as illustrated in Figure 2.

For each hemisphere:
 For each latitude:
  SymAsym: evaluate the appropriate signs of the spherical harmonics
  SpecToFreq: compute truncated expansions of the fields.
  MdFFTGrid: compute FFT of each truncated field.
  Vertig: compute nonlinear products by direct multiplications.
  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
 Next latitude
Next hemisphere

---

*Figure 2: Deeply nested sequential **For** loops in function **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 the function **Nonlinear**, is given in Figure 4. This subroutine performs a transformation to compute the truncated expansions of the pg, zg, ug and vg fields. The mappings resulted in two inner parallel **For** loops which performed summations, inside an outer sequential loop which updated the arrays of the four fields. With many other similar sequential codes , some more costly than **SpecToFreq**, residing inside another two sequential **For** loops (***For each hemisphere*** and ***For each***

latitude) as shown in Figure 2, the result was a large number of complicated sequential array updates in deeply nested sequential loops which led in turn to a large amount of array copying. The memory requirement for this implementation was many times larger than that for the FORTRAN implementation where update in place is possible.

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, implicit mappings or intrinsic complex number type. Additionally the *side effects* resulting from the use of **common** and **equivalence** statements made it difficult for the context of the model to be understood.

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 50% longer than the original FORTRAN code. The links between the mathematical formulation and both the FORTRAN and the directly transliterated SISAL codes were also obscure.

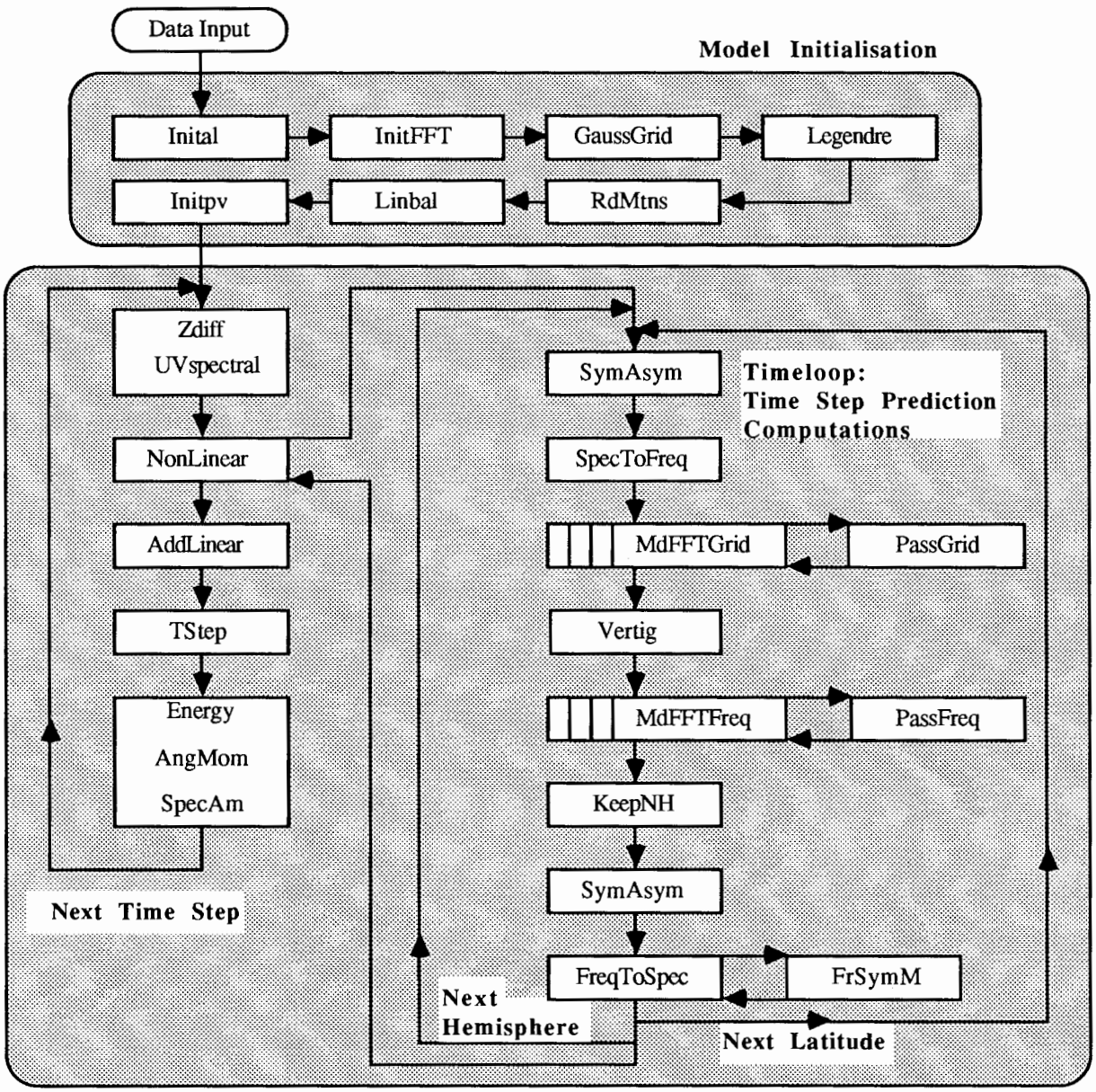

Figure 3: Flow chart for the FORTRAN and sequential SISAL implementations

```
      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

      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*

```
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;
        pgr, pgi, zgr, zgi :=
            IF old m = 1 & j = 1
                THEN 0.0, 0.0, 0.0, 0.0
                ELSE   alp[jmx] * pri[jmr],
                       alp[jmx] * pri[jmi],
                       alp[jmx] * zri[jmr],
                       alp[jmx] * zri[jmi]
            END IF
        RETURNS  VALUE of SUM pgr
                 VALUE of SUM pgi
                 VALUE of SUM zgr
                 VALUE of SUM zgi
        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*

2.3 Results for Direct Transliteration Approach

The SISAL and FORTRAN programs were run on an Encore Multimax with APC (32332/32081) processors. The results were obtained using the standard f77 FORTRAN compiler released with the Encore Multimax (Umax 4.2, release 3.3.0), and the optimising SISAL compiler (OSC) [3] from the Colorado State University. The FORTRAN compilations were performed with optimisation.

The model size chosen was a realistic spectral resolution with J = 30 (corresponding approximately to a 420 km grid). The run times of the model with one iteration of the timeloop at this resolution for multiple processors in Figure 5a shows that the SISAL run time on a single processor (773.0 seconds) was 11 times slower than the FORTRAN run time. The results with multiple processors sharing the workload indicated that this approach was very inefficient because the code iteratively progressed through the transformation section hemisphere by hemisphere, and in each hemisphere latitude by 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) as confirmed by the concurrency profile in Figure 5b.

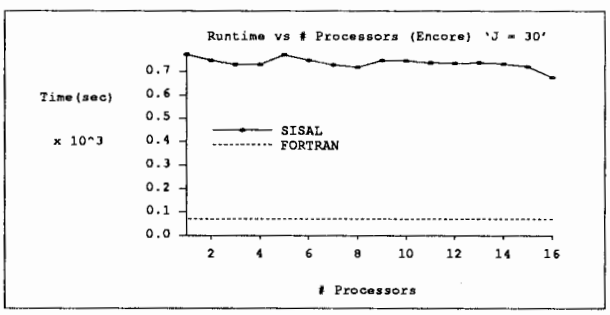

Figure 5a: Multiple processor run times for FORTRAN and sequential SISAL

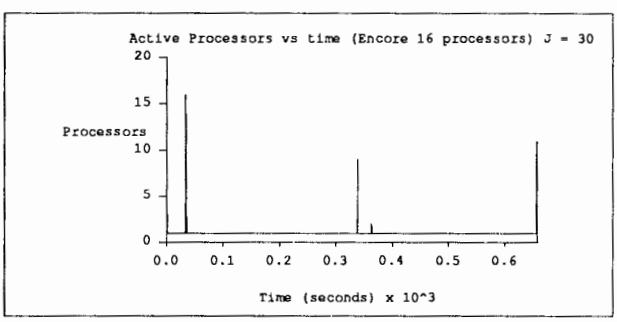

Figure 5b: Parallelism profile of the directly transliterated SISAL version

3. Parallel Implementation of the Model

Our analysis indicated that the new state of each field of interest in a current time frame was dependent on its values in the previous time frame, but the new values of the state were themselves independent of each other. By subsequent examination of the mathematical model it was seen that a parallel formulation of **Nonlinear** could be implemented as follows.

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

```
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;                  % SpecToFreq till here
   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 **SpecToFreq**, the truncated expansion computation*

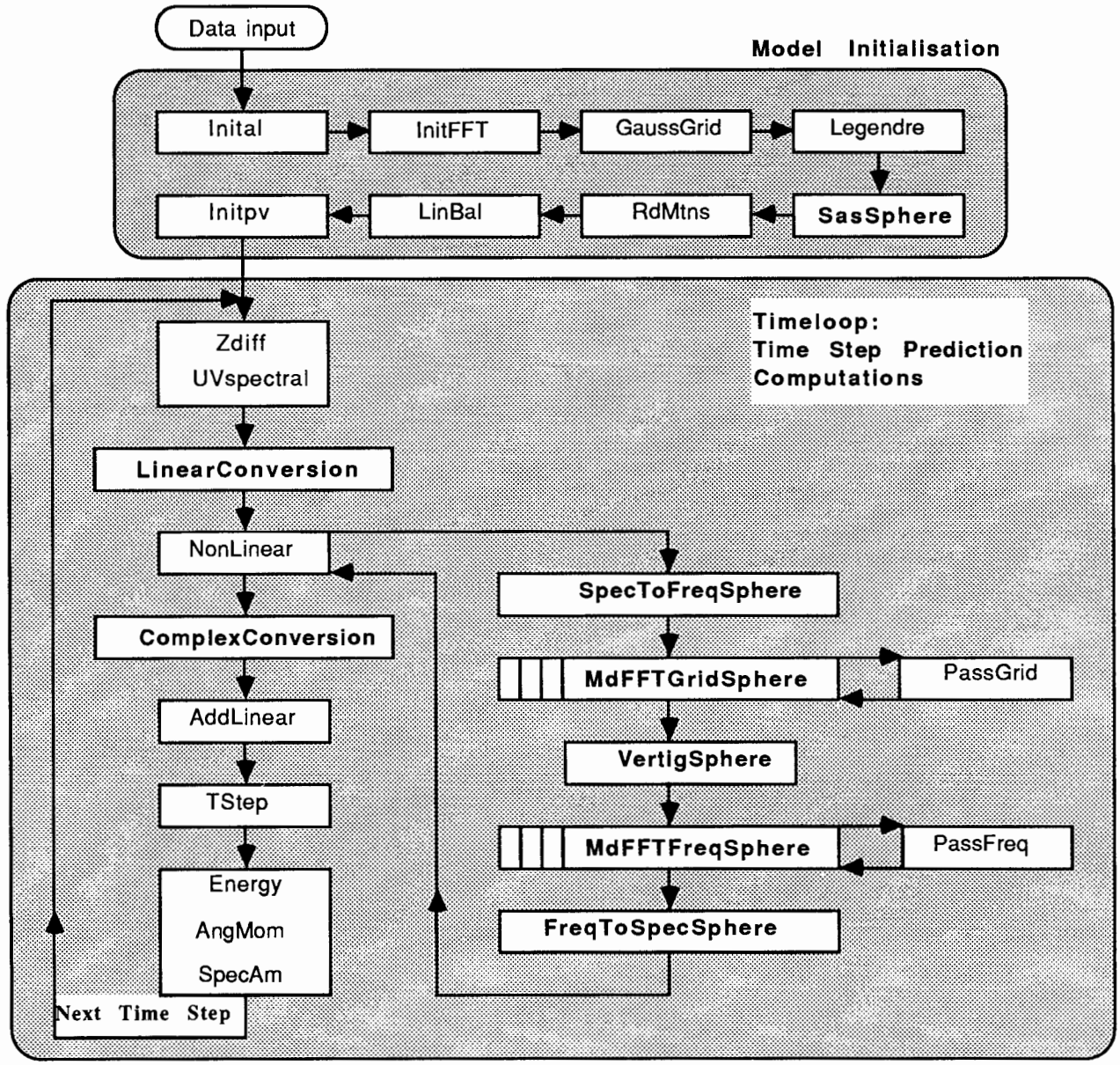

Figure 7: Flow chart for parallel SISAL implementation

This resulted 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 had been rewritten so that its outer loop also generated 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 has been implemented, with entirely new arrays being created directly using deeply nested parallel **FOR** loops instead of old arrays being updated in nested sequential loops. This produced more concise code and a memory requirement which was lower relative to that of the sequential implementation. Additionally the deeply nested parallel loop constructs enable OSC to slice the **FOR** loops more effectively.

The resulting parallel formulation in Figure 7 shows that evaluations of both the global symmetrical and anti-symmetrical spherical harmonics had been relocated so that they were precomputed concurrently only once in **SasSphere** in the model initialisation stage. They were stored as templates and were retrieved as required in the timeloop. The algorithm in each of the subroutines **SpecToFreqSphere**, **MdFFTGridSphere**,

VertigSphere, **MdFFTFreqSphere** and **FreqToSpecSphere** within **Nonlinear** had been rearranged as computation of the sphere in a whole "slice", and implemented using the SISAL deeply nested parallel **FOR** loop construct. All other subroutines in the timeloop also had their algorithms rearranged so that new arrays were created instead of old ones being updated. The resulting new codes were shorter and show a direct link to the corresponding mathematical equations.

Further improvement in speedup and concurrency has been obtained by our analysis of the serial code sections [4]. This included explicit slicing of singly nested loops containing small loop bodies. This was necessitated by the failure of the OSC cost estimator to recognise the critical path significance of a few routines containing these loops. Another aspect was the parallelisation of **Legendre**, a very costly routine in the initialisation section which computes the spherical harmonics for each latitude of the sphere 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.

3.1 Results for the Parallel Implementation

The run times of the model as a function of processors used for the model size J = 30 with one iteration of the timeloop as plotted in Figure 8a indicates that SISAL's parallel implementation (106.7 seconds) executes 7 times faster than its sequential implementation on a single Multimax processor. The plot shows additionally that with 16 processors sharing the workload, the run time for this model size has been reduced to 13.7 seconds.

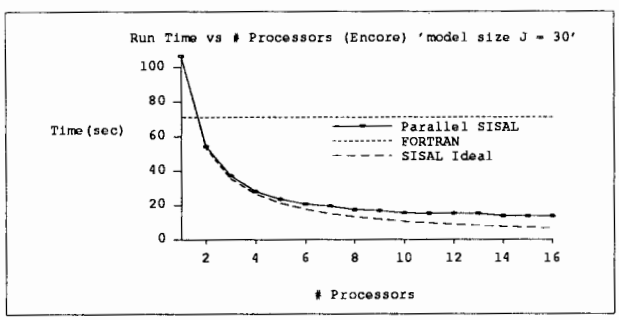

Figure 8a: Execution time profile of the new implementation

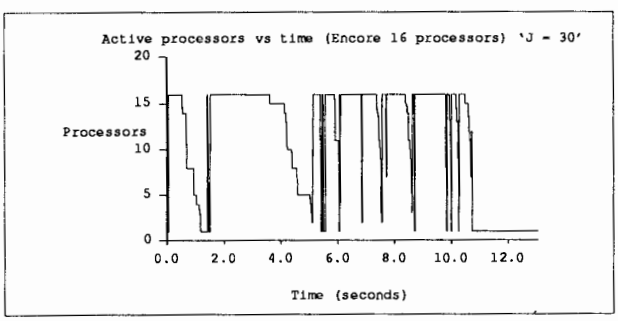

Figure 8b: Concurrency profile for the new implementation

The corresponding achieved parallelism in Figure 8b indicates that the parallelisation of the timeloop body, commencing at approximately 5.5 seconds, has been successful. 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 in its execution time.

4. Benchmarking of Timeloop

For a model size of J = 30, each time step is 30 minutes; hence 48 iterations of the timeloop are needed to perform a 24-hour forecast. The performance of the timeloop therefore is critical since it dominates the computation time of the model. It follows that one iteration of the timeloop is sufficient and adequate for the following benchmarking.

4.1 Performance in terms of Model Sizes

The 1 processor FORTRAN (F1), 1 processor SISAL (S1) and 16 processor SISAL (S16) execution time of the timeloop for varying model size are depicted in Figure 9. Similar to that observed by Bourke [1], the FORTRAN curve up to J = 30 is approximately proportional to J^2. The run times for SISAL beyond J = 30 are obtained from additional dummy data sets which still perform the same amount of computation, because the available datasets only allow the model size to be increased up to J = 30. The same datasets for FORTRAN are however difficult to arrange, and therefore the FORTRAN curve is extrapolated from J = 30 tangentially, thus conservatively. The plots show that the parallel implementation in SISAL with a single processor has an execution time curve very close to the sequential implementation in FORTRAN, though slightly slower. For SISAL running on multiple processors the growth in execution time with increasing model size is much slower than that for a single processor for either FORTRAN or SISAL.

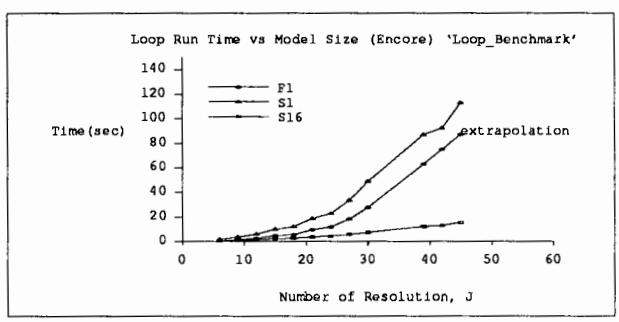

Figure 9: The execution time of the FORTRAN and SISAL implementations as a function of model size.

4.2 Speedups from the Benchmark Ratios

The above execution time benchmarks have been analysed and expressed as S1/F1, F1/S16 and S1/S16 speedup ratios as a function of model size as plotted in Figure 10. The S1/F1 curve refers to the speedup of the FORTRAN implementation over the SISAL implementation for varying model size. It indicates that as the model size is increased, the implementation in SISAL on conventional computer

systems, on equal terms, executes competitively or could even be faster than the FORTRAN implementation.

The F1/S16 curve refers to the speedup of the SISAL implementation in a multiprocessor environment where 16 processors share the workload simultaneously, over a sequential implementation in FORTRAN where a single processor performs the whole task. The positive gradient of the F1/S16 curve suggests that the speedup of the 16 processor SISAL execution time over the FORTRAN execution time can be even more substantial when a larger model size is adopted. The ratio justifies the use of a parallel implementation of the weather model.

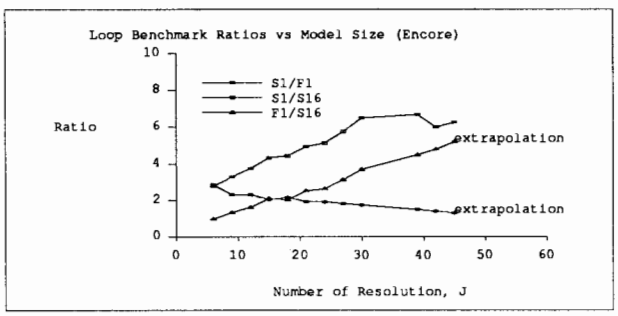

Figure 10: The benchmark ratios as a function of model size.

Finally, the S1/S16 curve refers to the speedup of the SISAL implementation in a multiprocessor environment, where 16 processors concurrently share the workload, over the execution time of the same task by a single processor. It shows that the larger the model size, the higher will be the achieved speedup because the level of parallelism is higher as illustrated in the concurrency profiles in Figures 12a and 12b; Figure 12b particularly shows that the relative time spent in sequential regions is reduced as excess available tasks in the SISAL run time task queue are transferred forward in time, thus extending the parallel regions. As well as showing the general principle of parallel processing, the S1/S16 curve also indicates that employing SISAL on multiprocessors can provide efficient speedup over a single processor on the Multimax multiprocessor. Interestingly, the falling gradient of this curve at the extended model sizes J = 39, 42 and 45 presents itself as a symptom of a very inefficient memory deallocation routine of the OSC run time system.

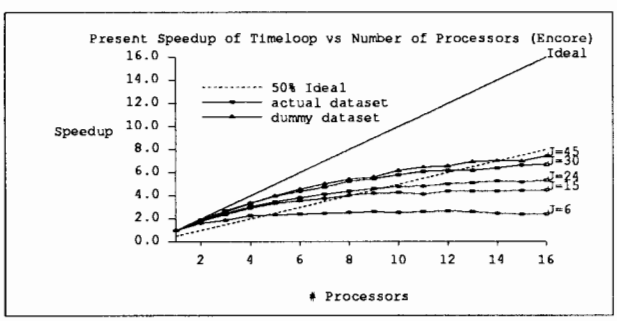

Figure 11: The speedup profile of the timeloop for the present implementation

This, in addition to other overheads contributed by SISAL, the OSC run time system, the hardware architecture and the algorithm, results in the S1/Sn speedup curves for various model sizes as plotted in Figure 11. The speedup curves indicate that as the number of processors is increased the speedup increases to a maximum value, and then decreases as the ratio of OSC run time overheads to useful computation increases [5][6]. This is particularly evident for small model sizes where the run time system is attempting to exploit a small amount of potentiak concurrency.

4.4 Effect of Memory Deallocation Overhead

Figures 12a and 12b also indicate that significance of the serial tail section becomes critical as more parallelism is obtained, thus contributing to the overhead shown in the S1/S16 ratio in Figure 11. This tail has been found to be produced by the eager memory deallocation routine of the OSC run time system in which the storage structures used are automatically but sequentially deallocated at the end of every cycle of the timeloop. As a result, the overhead consumes a large proportion of the loop time in Figure 12b (approximately 28%). This confirms the observations of Cann and Oldehoeft in their experience in running a SIMPLE hydrodynamics code on a Sequent Balance 21000 [3].

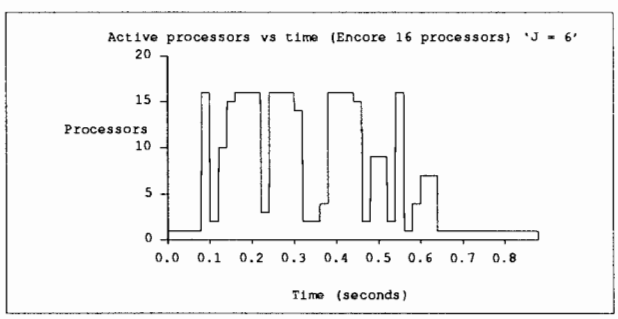

Figure 12a: Concurrency profile of timeloop for J = 6 (small model size)

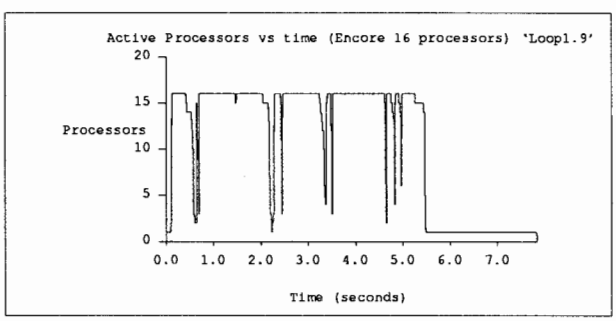

Figure 12b: Concurrency profile of timeloop for J = 30 (large model size)

From our analysis, better static analysis may determine that the array sizes are invariant through loop iterations and may be re-allocated. Also, "lazy" deallocation of memory structures in parallel with the main computation only when necessary would lead to substantial gains. While the issue of resource utilisation is being investigated by us and improved deallocation strategies are being further investigated and

implemented by the the developers of OSC, Figures 13a and 13b illustrate the achievable results that more efficient memory deallocation could have. In this example we removed the deallocation code entirely. The effect is a loop iteration with significantly improved speedup characteristics. With a fixed number of processors sharing the workload concurrently, 16 for example, the gradients of the speedup curves increase with model size. At the same time, the percentage machine utilisation, or efficiency also improves. These features indicate that more processors may be added, if necessary, to provide further speedup with good machine utilisation for larger model sizes.

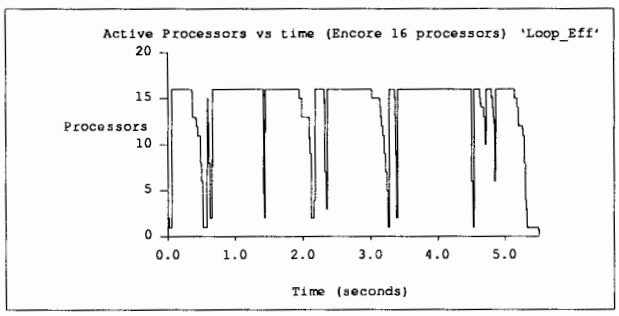

Figure 13a: The achievable concurrency (J = 30) of the timeloop with an efficient memory deallocation scheme.

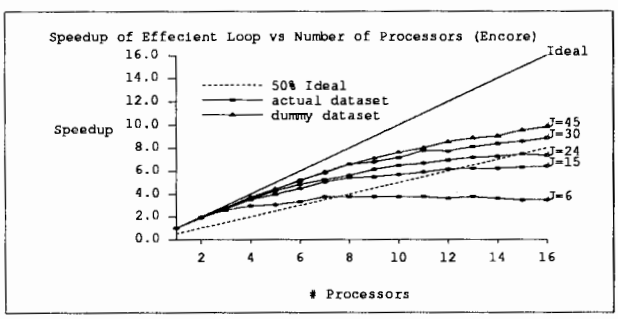

Figure 13b: The would be speedup of the timeloop with an efficient memory deallocation scheme.

5. Conclusions

The results of the study described in this paper confirm the feasibility of a parallel implementation of the adopted weather model in SISAL. They show that for larger model sizes, the SISAL single processor performance is gratifyingly close to that of FORTRAN on the Encore system while its multiprocessor performance is also encouraging eventhough our results show significant effects of hardware, software and OSC run time overheads. 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 [1][6]. Due to the implicit expression of concurrency with SISAL, at no time were we concerned with the underlying machine organisation.

Acknowledgements

We 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. We 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 Research Organisation (CSIRO) and the Royal Melbourne Institute of Technology.

References

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

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

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

[4] P.S. Chang and G.K. Egan, "Performance Evaluation of a Parallel Implementation of Spectral Barotropic Numerical Weather Prediction Model in the Functional Language SISAL", Technical Report TR 118 091 R, Department of Communication and Electrical Engineering, Royal Melbourne Institute of Technology, Oct. 1989.

[5] P.S. Chang et al, "Parallel Execution of a Barotropic Numerical Weather Prediction Model in SISAL on an Encore Multimax Multiprocessor", Technical Report TR 118 093 R, Department of Communication and Electrical Engineering, Royal Melbourne Institute of Technology, Dec. 1989.

[6] 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.

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

[8] I. Simmonds, "The Spectral Modeling Project", Report No. 1, August 1978, Department of Meteorology, University of Melbourne.