Performance Evaluation of a Parallel Implementation of Spectral Barotropic Numerical Weather Prediction Model in the Functional Dataflow Language SISAL

Pau S. Chang and Gregory K. Egan

Joint RMIT/CSIRO Parallel Systems Architecture Project
Department of Communication and Electrical Engineering
Royal Melbourne Institute of Technology
124 La Trobe St, Melbourne 3000,
Victoria, Australia

Abstract

A one-level barotropic spectral Numerical Weather Prediction (NWP) model has been implemented [4] in the high-level parallel functional dataflow language SISAL [3]. In furthering that work, our attention has been focused on identifying the sources of the sequential sections in the concurrency profile for the time-step section, and in parallelising a very significant serial code section in the initialisation section of the spectral model. The subsequent code refinements, the performance evaluation techniques, the cost of the automatic sequential memory deallocation scheme of the OSC runtime system, the solutions of the encountered problems and their effects, and the study of issues related to the effective use of current and next generation multiprocessors raised as a result of the analysis are discussed in this paper.

1 Introduction

Numerical Weather Prediction (NWP) is acknowledged as being of vital importance to the Australian and world economies. The demand that NWP places on computing system performance has increased dramatically since the introduction of computer systems. 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 amount of inherent parallelism, in many cases this has been lost in the formulation of sequential and vector systems; additionally there is a strong suggestion that existing language systems will prove inadequate for the new multiprocessors.

In our earlier work [4] on feasibility research, we implemented a one-level barotropic spectral NWP model [1] using the high-level parallel functional dataflow language SISAL (Streams and Iteration in a Single Assignment Language) [3]. The analysis of the initial results which were obtained using an Optimising SISAL Compiler (OSC) [2] for a shared-memory ENCORE Multimax with 16 APC (32332/32081) processors and a SUN 3/260 (68020/68881), leads to the continued study of issues related to the effective use of current and next generation multiprocessors [6].

In furthering the ealier work, our attention has been focused on identifying the sources of the sequential sections on the concurrency profile of the time-step section (Figure 2), and in parallelising a very significant serial code section which evaluates Legendre polynomials in the initialisation stage of the full spectral model (Figure 1). The research has indicated that the current fixed global parallel execution cost estimator in OSC may lead to unsatisfactory results in some cases. The subsequent code refinements, the resulting characteristics of the implementation, the performance evaluation techniques and the effect of the sequential memory allocation routine will also be discussed.

The detailed mathematical expressions and descriptions of this NWP model can be found in [1]. In this paper, the model size is expressed in terms of its *resolution number* J. The *spectral truncation limits* jx, jxx and mx are related to J where jx = mx = J + 1 and jxx = J + 2. The *number of latitudes* of the globe and the *number of longitudinal* points on each latitude are also related to J by ilat = (5 * J + 1) / 2 and ilong = 3 * J + 1 respectively.

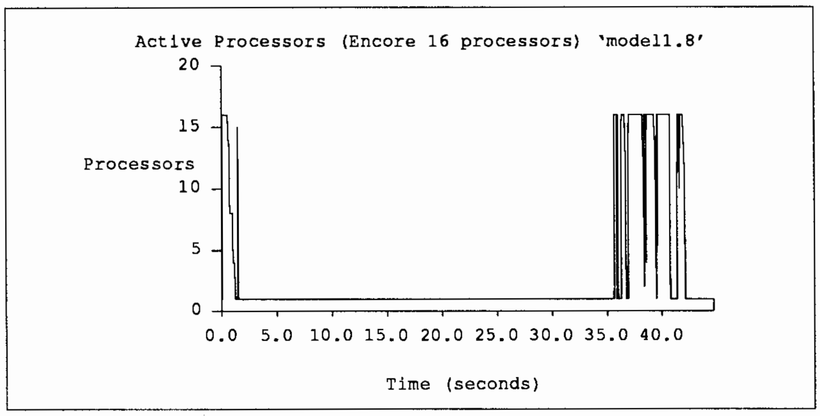

Figure 1: The concurrency profile of the full model from the implementation in [4]

2 Sequential Code Sections in The TimeLoop

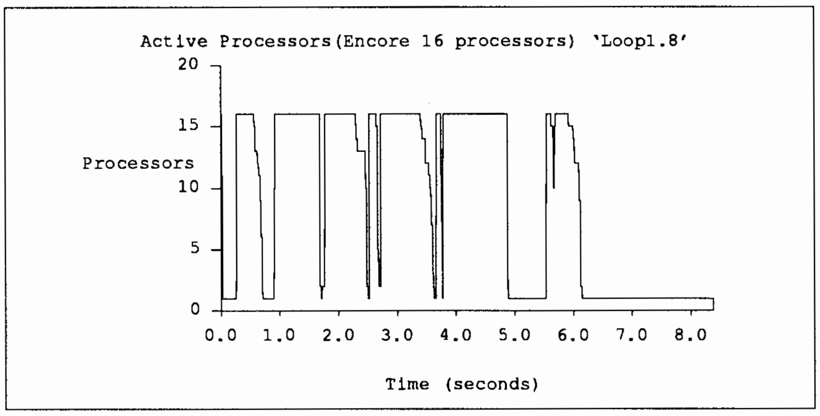

Figure 2: The concurrency profile of the timeloop extracted from the model

The parallelism profile of the initial SISAL implementation for a model size of J = 30 with 16 processors sharing the workload (Figures 2) indicates that the inter-function sequential or Amdahl [7] notches in the timeloop caused by data dependency have a second order effect on potential speedup. There still remain however three significant serial sections which consume approximately 13% of the total execution time and grow with problem size.

The sequential sections are due to three specific functions, all of which are involved in building single dimensional arrays from single nested parallel **FOR** loops which contain small loop bodies. A number of subsequent experiments have shown that the OSC's slicing and parallelisation of small body single nested parallel **FOR** loops which produce single dimensional arrays are globally determined by the compile time routine which estimates the parallel execution costs, and these loops are not sliced due to the failure of the OSC cost estimator to recognise the critical path significance of these functions. Discussions of a similar issue, but for a dataflow architecture, on the relationship between the body of a parallel **FOR** loop and the exploitable concurrency residing in it can also be found in [5].

2.1 Single Nested Parallel Loop Involving a Simple Loop Body

SISAL presently does not implicitly support the data structure for complex numbers. Hence a complex number is represented by a **record** of two numbers, *Repart* and *Impart* in our implementation. Figure 3a shows one of these functions which explicitly converts four arrays of real numbers, each having an array size of jx * mx * 2, to four corresponding arrays of complex numbers of array size jx * mx each. Regardless of the number of processors used, the total length of the serial code on the concurrency profile undesirably increases with the size of the loop bound.

Nevertheless, this code can be locally compiled with the maximum slicing of the parallel **FOR** loop enforced by using the -H1 pragma of OSC. The Local Maximum Slicing or LMS curves in Figures 4a and 4b illustrate the desired improvement to this code as a result. However, the present OSC does not support the linkage to a seperately compiled routine, and therefore the amount of slicing of parallel **FOR** loops can only be specified as a globally effective OSC pragma at compile time. Unfortunately a pragma value of -H1 results in slicing of the routines of interest but leads to over-parallelisation of the rest of the program [2]. This in turn results in an execution time of 30 seconds compared with the original 8 seconds for the model size J = 30.

```
ctC, eC, ptC, ztC :=
   FOR complex_index IN 1, jxmx
    index := complex_index * 2
   RETURNS  ARRAY of RECORD CplexReal[Repart : ct[index - 1]; Impart : ct[index]]
            ARRAY of RECORD CplexReal[Repart : e[index - 1]; Impart : e[index]]
            ARRAY of RECORD CplexReal[Repart : pt[index - 1]; Impart : pt[index]]
            ARRAY of RECORD CplexReal[Repart : zt[index - 1]; Impart : zt[index]]
   END FOR
```

*Figure 3a. A single nested parallel **FOR** loop with a small loop body*

```
ctC, eC, ptC, ztC :=
    FOR m IN 1, mx
    ctC, eC, ptC, ztC :=
          FOR j IN 1, jx
          complex_index := jx * (m - 1) + j;
          index := complex_index * 2
          RETURNS ARRAY of RECORD CplexReal[Repart : ct[index - 1]; Impart : ct[index]]
                  ARRAY of RECORD CplexReal[Repart : e[index - 1]; Impart : e[index]]
                  ARRAY of RECORD CplexReal[Repart : pt[index - 1]; Impart : pt[index]]
                  ARRAY of RECORD CplexReal[Repart : zt[index - 1]; Impart : zt[index]]
          END FOR
    RETURNS VALUE of CATENATE ctC
            VALUE of CATENATE eC
            VALUE of CATENATE ptC
            VALUE of CATENATE ztC
    END FOR
```

Figure 3b: A quasi double nested loop

A possible solution to this problem is to augment the cost estimation of OSC by providing the number of processors available on the target machine as an additional pragma. The likely effect of this solution can be demonstrated by explicitly slicing these loops using a *quasi double nested* (QDN) technique, which returns in this case the desired single dimensional arrays. The technique may be used effectively to force appropriate decisions from the current OSC cost estimator.

2.2 *Quasi Double Nested* Technique

Using the QDN technique, as shown in Figure 3b, the inner parallel **FOR** loop of loop bound jx computes the correct array indices. This loop resides inside an outer parallel **FOR** loop, of loop bound mx, which concatenates every temporary array it produces. The loop body of this outer loop hence becomes larger and an order of complexity higher. This technique produces a slightly larger code size but the overhead is felt only when one processor is employed. Furthermore, the execution time and concurrency profiles produced using this technique are the same as those produced when the original code is locally compiled with maximum slicing, as already discussed in the previous section. Figures 4a and 4b illustrates the consequent efficient exploitation of concurrency and an execution time of 3.5 times faster for this code relative to the original unparallelised code.

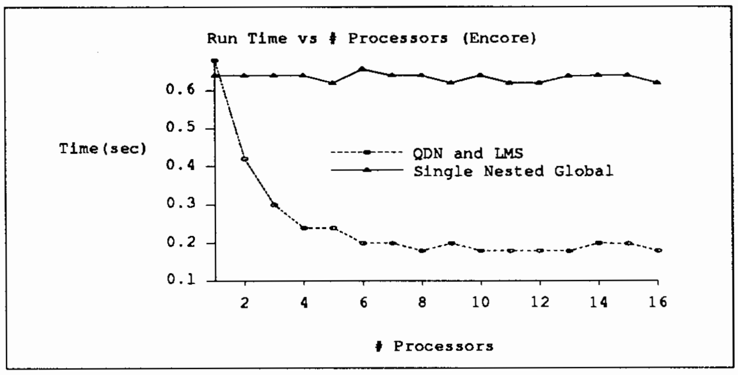

Figure 4a: The comparison of execution time as a function of the number of processors (J = 30)

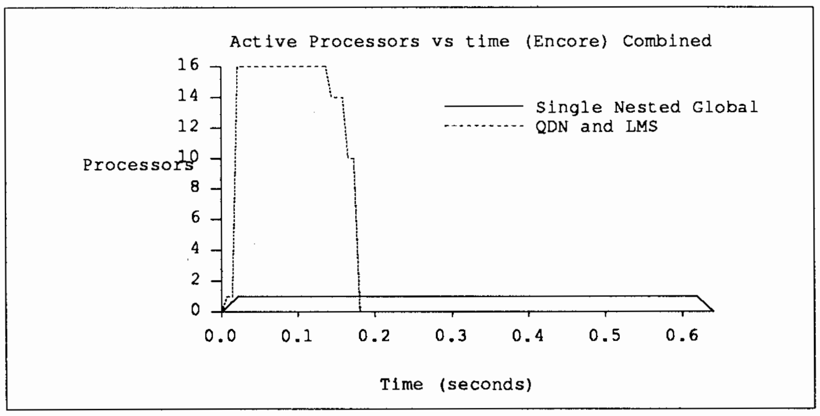

Figure 4b: The comparison of concurrency profiles as a function of the number of processors (J = 30)

3 Concurrency in the Computation of Spherical Harmonics

The serial computations for the Legendre polynomial of the first kind which produce the spherical harmonics of the globe [1] dominate the initialisation stages of both the FORTRAN and the initial SISAL models. These sequential computations when parallelised significantly speed up the initialisation stage.

3.1 Serial Implementation

The model groups the latitudes of the globe into ilat / 2 number of North-South latitude pairs. For each latitude pair, the function **LEGENDRE** computes jxx * mx number of spherical hamonics. The conventional implementation is sequentially conceived where both the computations for each of the harmonics on the same latitude pair and for each frame of latitude pairs are executed sequentially. In other words, there are a total of ilat / 2 * jxx * mx harmonics produced and hence the same number of corresponding serial array updates performed. The SISAL equivalent of the FORTRAN version, from direct transiteration, is shown in Figure 5a.

```
alp := FOR INITIAL
       WORKlgn  := ARRAY_fill(1, jxxmx, 0.0d0);
       lat_level := 1;
       alp_LGN  := LEGENDRE(ir, irmax2, jxxmx, coaiy[1], siaiy[1], deltaiy[1], WORKlgn);
       WHILE lat_level < ilat / 2 REPEAT
       lat_level := old lat_level + 1;
       alp_LGN  := LEGENDRE(ir, irmax2, jxxmx, coaiy[lat_level], siaiy[lat_level],
                            deltaiy[lat_level], old alp_LGN);
       RETURNS ARRAY of alp_LGN
       END FOR
```

Figure 5a: A sequential computation of the spherical harmonics

```
alp := FOR lat_level IN 1, ilat / 2
       alp_LGN  := LEGENDRE(ir, irmax2, jxxmx, coaiy[lat_level], siaiy[lat_level],
                                        deltaiy[lat_level]);
       RETURNS ARRAY of alp_LGN
       END FOR
```

Figure 5b: A parallel computation of the spherical harmonics

3.2 Parallel Implementation

The above SISAL routine can be conveniently parallelised using the SISAL forall construct (Figure 5b). In this version, all frames of latitude pairs (**LEGENDRE**) are computed concurrently because they are found to be data independent of each other. Figure 6a and 6b show the dramatic improvement in the run time and concurrency of the initialisation stage of the model for J = 30.

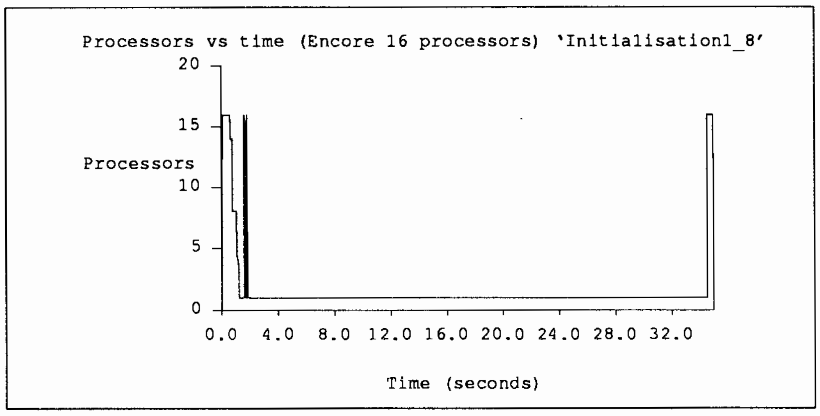

Figure 6a: The concurrency profile of the sequential SISAL implementation

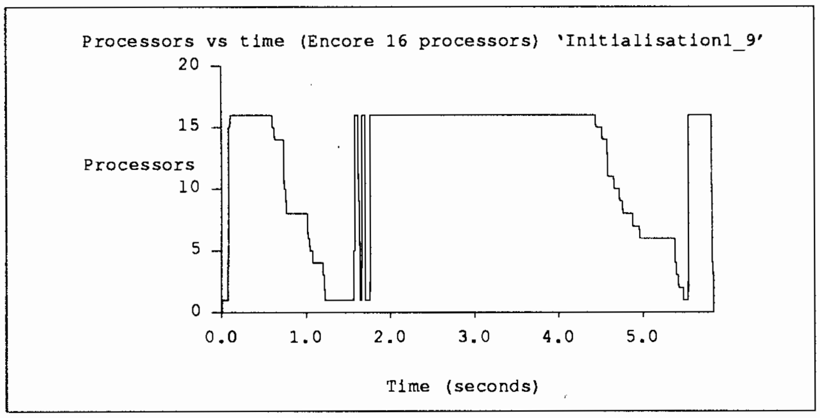

Figure 6b: The concurrency profile of the parallel SISAL implementation

The improved performance over the initial implementation as illustrated in Figure 6b suggests that all processors have been kept busy throughout. **LEGENDRE** could be coded as a "wavefront" algorithm leading to additional improvement however, due to the satisfactory gains already obtained, this is not presently being pursued.

4 Characteristics of the New SISAL Model

The available dataset only allows the model size to be increased up to J = 30. In order to show the capability of the new SISAL implemention in the exploitation of concurrency of larger model size, and its potential in providing substantial speedup over the sequential FORTRAN version, J has been extended beyond 30 by building additional dummy datasets which still perform the same amount of computation.

4.1 Execution Time versus Number of Processors Profile

Figures 7a, 7b and 8 illustrates the performance of the refined implementation. The execution time profile (Figure 7a) indicates that the runtime of a small model size saturates quickly with increasing number of processors because there is not enough available parallelism to be exploited. However, when the model size grows larger, the increased available concurrency lowers the rate of saturation. The "Ideal" line shown assumes two unrealistic conditions that the SISAL code is perfectly parallel and the ENCORE Multimax architecture is fully capable of exploiting this parallelism. It is nonetheless included to show that the actual run time for model size J = 30 approaches ideal.

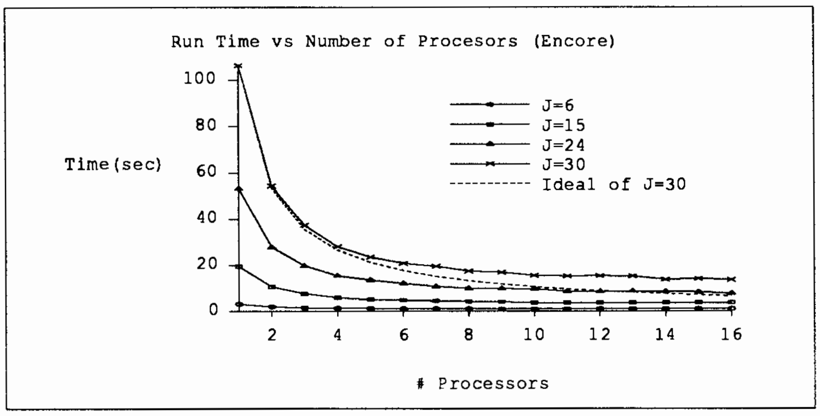

Figure 7a: The model execution time profile

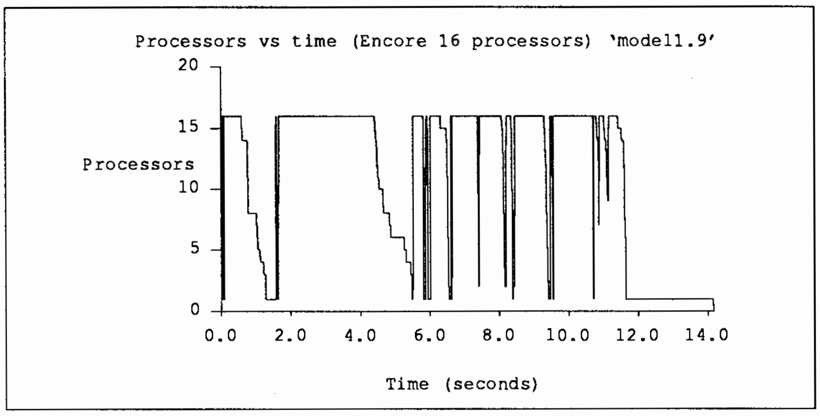

Figure 7b: The concurrency of the model for J = 30

4.2 Speedup versus Number Of Processors Profile

When varying number of processors are employed concurrently, the achieved speedups over single processor execution time for various model sizes are shown in Figure 8. The "Ideal" line, which assumes the same conditions as Figure 7, in this case indicates 100% machine utilisation.

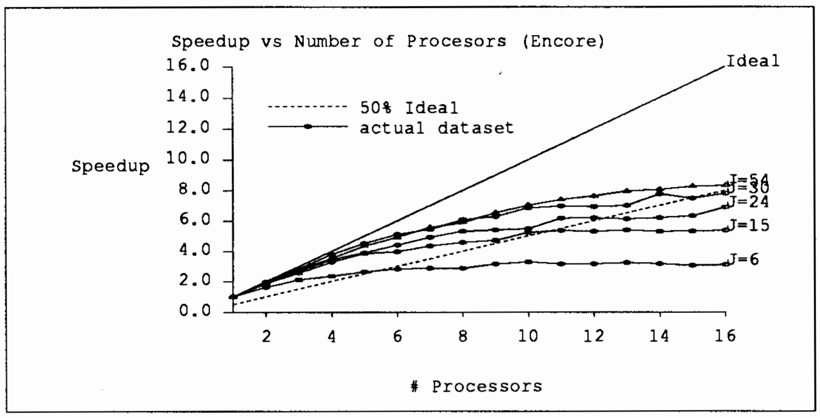

Figure 8: The model speedup profile

With a fixed number of processors, 16 for example, the gradients of the speedup curves increase with model size. At the same time, the percentage machine utilisation 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.

4.3 Benchmark Against the FORTRAN Implementation

The FORTRAN and the new SISAL Barotropic model implementations generally consist of an initialisation section in which all lookup tables and the initial values of weather variables are sorted, and a timeloop section in which the future weather states are computed. For a 24 hour forecast using the model size J = 30, the models will need to iterate 48 times in the timeloop, which therefore dominates the computation. Hence, benchmarking of the timeloop with one iteration is sufficient.

4.3.1 Execution Time Benchmarks

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. 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. The curves are in general a function of J^3. The second derivative of the S16 curve is however very much smaller relative to that of the F1 curve. Alternatively stated this means a much slower growth in execution time with increasing model size for SISAL.

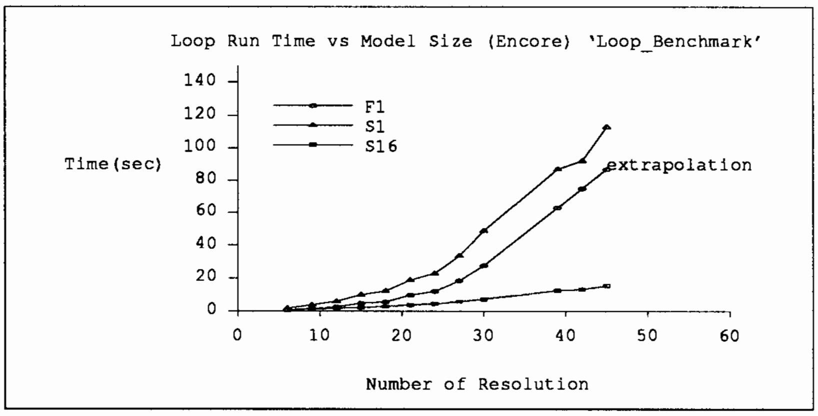

Figure 9: The execution time of the FORTRAN and SISAL implemetations as a function of model size. F1 curve beyond J = 30 is an extrapolation.

4.3.2 Speedups from the Benchmark Ratios

The above execution time benchmark, when summarised using benchmark ratios, supports the potential of the new SISAL implementation in providing competitive execution times in conventional computer systems, and a scalable speedup in a multiprocessor environment. The ratios S1/F1, F1/S16 and S1/S16 are expressed as a function of model size (Figure 10) and are discussed in turn below.

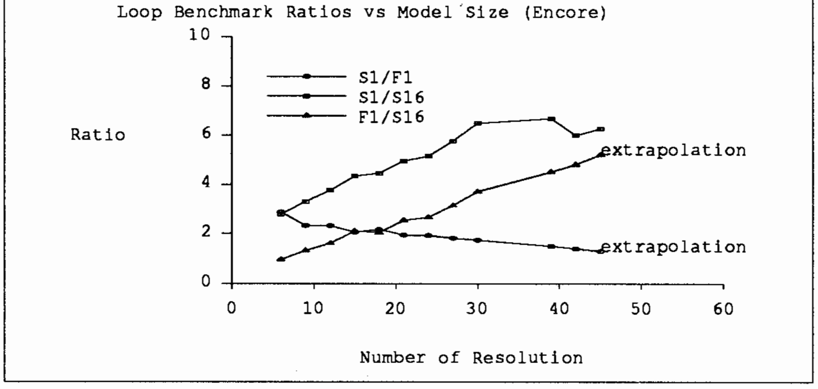

Figure 10: The benchmark ratios as a function of model size.

First, the S1/F1 curve refers to the speedup of the FORTRAN implementation over the SISAL implementation for varying model size. It indicates that the implementation in the functional dataflow language SISAL, on equal terms, will approach or perhaps may even execute faster than the FORTRAN implementation as the model size is increased.

Second, the F1/S16 curve refers to the speedup of the SISAL implementation in a multiprocessor environment where 16 processors share the workload in parallel, 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 better when a larger model size is adopted.

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 better will be the acheived speedup because the level of parallelism is higher. As well as showing the general principle of parallel processing, this ratio 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, a topic which will be discussed in the next section.

5 Effect of Automatic Memory Deallocation

Figures 11a and 11b illustrate the performance of the timeloop of the present implementation. The concurrency profile shows a significant section of serial code at the end of each time step in the timeloop, in the form of a long 'tail'. A number of experiments have indicated that this tail is 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. This confirms the observations of Cann and Oldehoeft in their experience in running a SIMPLE hydrodynamics code on a Sequent Balance 21000 [2].

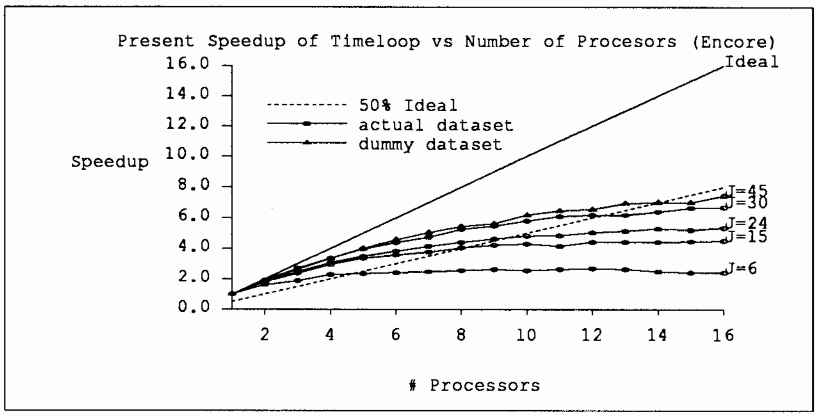

Figure 11a: The speedup profile of the timeloop for the present implementation

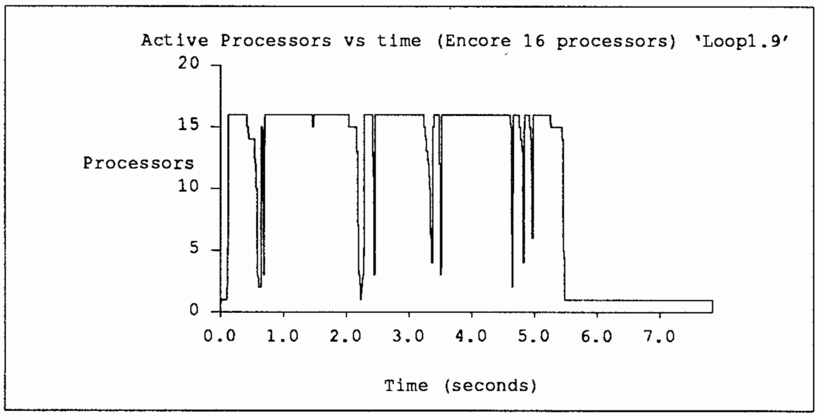

Figure 11b: The concurrency profile of the timeloop (J = 30) for the present implementation

In our case, the ENCORE and the SUN versions of this routine are both very inefficient, particularly the former which consumes a large proportion of the loop time (25% when using 16 processors concurrently). The effect of this tail is especially drastic when more parallelism is enhanced by larger model size, as has already been shown in the S1/S16 curve in Figure 10.

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. We are currently investigating improved deallocation strategies.

Figures 12a and 12b illustrates the effect that more efficient memory deallocation could have. In this example we removed the deallocation code entirely. The result is a loop iteration with significantly improved speedup characteristics.

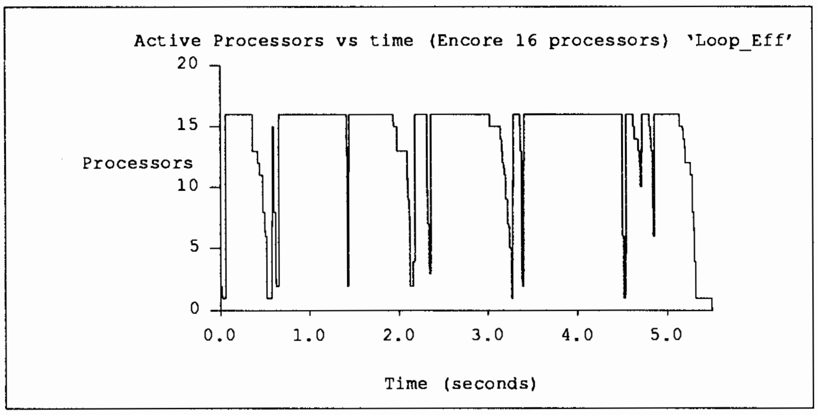

Figure 12a: The would be concurrency (J = 30) of the timeloop with an efficient implementation

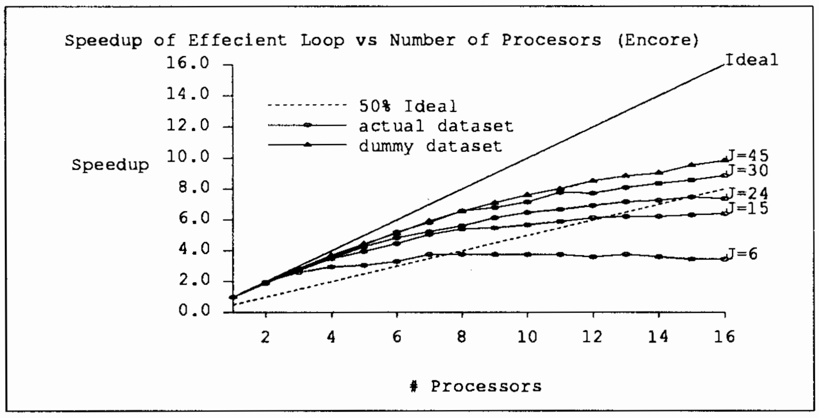

Figure 12b: The would be speedup of the timeloop with an efficient implementation

6 Conclusion

More accurate Barotropic NWP modeling requires the use of a larger model size. Apart from a large memory requirement, weather modeling researchers are hampered in adopting large model sizes because of the associated $O(J^3)$ growth in execution time on conventional uniprocessors. We have demonstrated that the problem of the long execution time can be reduced by a parallel implementation of this weather model in the functional dataflow language SISAL on a general purpose shared-memory multiprocessor.

We have identified that the current global parallel execution cost estimation routine of OSC, and its eager and sequential memory dellocation scheme can lead to unsatisfactory results in some circumstances. We have suggested solutions and demonstrated the effect these solutions have on execution times. We have shown that the resulting improved SISAL implementation exhibits good scalability for very large model sizes on large multiprocessor systems. This research will lead to further work by ourselves and the OSC developers in the implementation of a more appropriate memory deallocation scheme and parallel execution cost estimation function.

7 Acknowledgements

We would like to thank Drs Ian Simmonds and Ian Smith of the Department of Meteorology at the University of Melbourne for providing access to, and interpretation of, the original FORTRAN implementation of the Barotropic Spectral model. We thank Warwick Heath for his work on instrumentation. Our gratitude is also extended to all members of the Project team, particularly Dr David Abramson and Neil Webb, for their contributions to the discussions on memory allocation routine. The RMIT/CSIRO Parallel Systems Architecture Project is jointly funded by the Royal Melbourne Institute of Technology and the Commonwealth Scientific and Industrial Research Organisation (CSIRO).

References

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

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

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

[4] P. S. Chang and G. K. Egan, "An Implementation of a Barotropic Spectral Numerical Weather Prediction Model in the Functional Dataflow Language SISAL", August 1989, submitted to the Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP) to be held in Seattle, Washington, on March 15-16, 1990.

[5] G. K. Egan, N. J. Webb and W. Bohm, "Numerical Examples on the RMIT/CSIRO Dataflow Machine", Technical Report TR 118 082 R, Joint RMIT/CSIRO Parallel Systems Architecture Project, RMIT, May 1989.

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

[7] G. Amdahl, "Validity of the Single-Processor Approach to Achieving Large-Scale Computer Capabilities", AFIPS Conference Proceeding, 1967, pp. 483-485.