Parallelizing CPTEC's General Circulation Model

Jonas N. Tamaoki[1], José Paulo Bonatti[2], Jairo Panetta[2], Simone Tomita[2]

[1] NEC do Brasil
Rod. Presidente Dutra km 40, Cachoeira Paulista, SP, Brazil
{tamaoki@cptec.inpe.br}
[2] INPE/CPTEC
Rod. Presidente Dutra km 40, Cachoeira Paulista, SP, Brazil
{bonatti, panetta, tomita@cptec.inpe.br}

Abstract—
We describe the first parallel version of CPTEC's General Circulation Model, targeting a 4 processor, shared memory NEC SX4. This paper emphasizes techniques to parallelize vintage production code, keeping results reproducible. Measured speed-ups compare favorably with Amdahl's Law predicted values.

Keywords— **Weather Forecast, Parallel Applications.**

I. Background and Motivation

The General Circulation Model (GCM) is the program that generates the daily global weather and the seasonal climate forecasts of the Brazilian Center for Weather Prediction and Climate Studies (CPTEC) of the National Institute of Space Research (INPE).

CPTEC's GCM has its roots on COLA's (Center for Ocean Land and Atmosphere Studies) GCM. A late 1980's cooperation agreement granted CPTEC access to the source code and the technology for its usage and modification. The code was ported to a single processor NEC SX3 by early 1990. It took around four years to build the software infrastructure (data acquisition, pre-processing, post-processing, product generation and delivery) to run GCM on production mode, as well as to transform GCM into a production program (including the insertion of triangular truncation). To estimate this effort, it suffices to state that the infrastructure and the GCM have around the same number of source code lines (31000 each).

Since November 1994 the model and the infrastructure have been successfully used for daily production runs at CPTEC. Around July 1998, CPTEC updated its computer facility to a 4 processor NEC SX4, sparking a model modernization project that requires the parallelization of GCM.

The model modernization project will produce a new version of GCM, with deep modifications on its mathematical aspects (e.g., from an Eulerian model to a Semi-Lagrangean model), software technology aspects (from pure procedural to semi-object oriented) and computational model aspects (from sequential to clusters of shared memory parallel machines). The first product of this enterprise is a shared memory parallel version of GCM, maintaining the original mathematical and software technology characteristics.

This paper describes techniques used to generate this first product and some of the parallelizing issues involved.

II. Characterizing the Computation

A meteorological description of GCM can be found at [BON 96] and at [KIN 97]. Our description is limited to the computational aspects required for the understanding of the parallelization procedure.

The forecast program advances through discrete forecast time. At each time step it computes the state of the atmosphere by updating a set of "fields" (arrays of floating point numbers) that store values of the physical variables (pressure, temperature, horizontal vector velocity, humidity) represented in the mathematical model. Updates occur by integrating a set of partial differential equations. At each time step, the integration is performed in two spaces:

1. Linear terms are computed at a three dimensional space denominated "spectral" space;
2. Non-linear terms are computed at a three dimensional space denominated "gaussian" space;

At each time step all fields have to be transformed from one space to the other and back to the original space. Computation starts and ends at spectral space.

Gaussian space has components latitude, longitude and vertical layers. Spectral space components are Fourier wave numbers, associated Legendre Functions and vertical layers. Consequently, at each time step, the fields at each vertical column of the atmosphere are represented in latitude - longitude coordinates and in wave number - Legendre Function coordinates.

In gaussian space, computation at each latitude is independent of all other latitudes. In spectral space, the computation at each vertical column is independent of all other vertical columns. The transformations from gaussian to spectral and back to gaussian are independent in the vertical layers. By dispatching independent computations to processors, it is conceptually easy to transform the sequential computation into a parallel one.

III. Characterizing the Program

What prevents the easy parallelization described above is the way the sequential program is built. It was designed to minimize memory usage, with no consideration to parallelization. That was achieved by the use of two techniques:

1. A field dimension is dropped whenever computations in that dimension are independent and results can be stored at another representation;
2. Scratch variables use a single memory area, whenever possible.

The first technique reduces memory requirements by eliminating one dimension from large arrays, while the second technique speeds up program execution and reduces scratch areas. The first technique prevents immediate program parallelization, since independent computations update the same memory area. The second technique greatly complicates program modification, a requirement for parallelization. Consequently, the program has to be heavily adapted prior to parallelization.

Besides, the sequential program is a moving target, since it is continuously updated by meteorological research results.

The sequential program consists of 31256 lines of very old style Fortran 77 (e.g., arithmetic ifs) distributed over 161 procedures, with 51 common blocks, each typically 20 arrays wide.

Summarizing, the conceptually easy parallelization is a programmer's challenge, since the sequential program is a moving target that was not designed for such an enterprise.

IV. Parallelization Strategy

A performance evaluation tool (NEC's Analyzer) [NEC 96] was used for time profiling the sequential program. Results are that 90.4% of the execution time is spent in just two loops. Both loops sweep latitudes, at a fixed time step, conveying the following computation:

1. Obtain field values at current latitude and time step, from their spectral space representation;
2. Compute the non-linear contribution (or physical processes) for this particular latitude;
3. Convert this contribution into spectral space, partially updating next time step field values.

Since each latitude computation is independent of all other latitudes, the natural parallelization is to execute, in parallel, loop iterations. The single critical issue in this approach is next time step field updating in spectral space, since two processes cannot update, simultaneously, the same field. The theoretical speed-up of this approach, estimated by Amdhal's Law, was considered acceptable for a first version and is described on table I:

TABLE I
Estimated SpeedUps

Processors	*Speed-up*
2	1.82
3	2.51
4	3.10

Two parallel models were considered for implementation: the parallel programming library approach (MPI/PVM style) and the compiler directives approach (OPENMP style). Due to the small number of processors, the shared memory programming model of the NEC SX4 and the software complexity of the sequential code, the compiler directives approach was selected

Fortran 77 compiler directives for program parallelization on the NEC SX4 use the microtask concept [NEC 97]. A set of microtasks (POSIX pthreads) is created at program initialization, and remains alive during program execution. Work quantum is a single loop iteration. These iterations are dynamically dispatched to the tasks. All tasks synchronize at loop initialization and termination. Directive syntax for a parallel loop is:

```
*pdir pardo
    do lat = 1, latMax
      .....
    end do
```

It is the programmer's responsibility to guarantee the independence of loop iterations. The compiler does not check for simultaneous updating of a single memory location by multiple microtasks (races). There are directives to avoid simultaneous updates, based upon the critical section concept.

The difficulty is that both loops encompass about 60% of all program lines, all commons included.

V. Parallelization Realization

Since iterations of the target loops are conceptually independent (pairs of iterations operate over distinct latitudes) the single obstacle is the need to avoid races for memory positions. That requires inspection, and possibly modification, of large portions of the code.

A. *Memory References*

An in-depth examination of common block arrays referenced in the parallel loop, based upon compiler produced cross-reference listings, classified each common block array in one of four cases, according to the access pattern within the loop:

1. Input data (arrays are used but not modified inside the loop);
2. Input/Output data (arrays are used prior to their modification and results are used outside the loop);
3. Output data (arrays are only modified within the loop and used outside the loop);

4. Scratch areas (remaining cases);

No modification is required for arrays of the first case.

The second case has to be carefully treated. Fortunately, only the update of the spectral representations of the fields falls in this case, since the gaussian field computed at a given latitude contributes to all wave numbers and Legendre Functions components of the spectral representation of the same field. Modifications required by this case are described at section VI-A.

Arrays at the third case are memory areas that do not overlap across loop iterations. They are multi dimensional arrays with the loop index as their last component. Consequently, nothing has to be done.

The fourth case represents memory areas that have to be replicated over concurrently executing loop iterations.

NEC provides a mechanism to deal with common areas on multi thread executions: common blocks can be labeled "global" (the same memory area is visible for all threads) or "local" (each thread creates its own copy of the common area). The syntax is to prefix the common statement with "local" or "global". Observe that these labels apply to a full common block, and not to a particular array in one common block. As one could expect, some original common blocks had arrays of the two kinds. These had to be split in two common blocks, one labeled 'global" and the other "local".

B. Programming Practices

The coding practice that required attention was data initialization inside the parallel loop. These were constructions of the form

if** (first) **then

that occur in three distinct cases, according to the use of the construct:

1. Used to initialize values that were latitude independent but time step dependent;
2. Used to initialize values that were latitude dependent but time step independent;
3. Used for specific latitudes (first and last, for example), of all time steps.

Constructions of the first case were modified so that each processor initializes its local copy of the data while processing its local first latitude at each time step. The second case was kept as in the sequential program, since each processor initializes its own copy of the data. The third case coding was modified to make the dependency on specific latitude values explicit.

C. Program Modification Cycle

The usual program development cycle (modify, compile, execute, verify results) was heavily mechanized. A tool to label and partition common blocks (into local or global) was developed, using UNIX utility *awk*, and heavily used. The tool allows rapid program modification, avoiding costly editing mistakes. It takes as input a list of common blocks, their labels, and a list of procedures where they should be labeled "local". Global is the default label for a common block.

Execution correctness was verified by a "checksum" technique. A sum of all elements of each output array was computed. Computed values were checked against corresponding values obtained during the sequential execution.

The utility of these two tools should not be underestimated. Even with such tools, it took eight man-months to produce an acceptable parallel version. It is also important to realize that GCM is continuously updated by meteorological research results. Consequently, modifications performed on early versions had to be executed again, on newer version of GCM.

VI. Results

A. Correctness

The parallel GCM produces correct results. The validation process includes comparing the forecasted fields generated by the parallel and the sequential GCM, with the same input data set. Visually, both fields are identical. Consequently, the parallel GCM was accepted for meteorological use. But, computationally, results differ on the eight significant digit (out of 14).

The fact that the results of the sequential version differ from those of the parallel version may indicate an error in the parallelization process. But oddly, the parallel results are the same, bit by bit, with any number of processes. Even with a single process.

We briefly discuss the single reason for this odd behavior – the parallelization of the gaussian to spectral transformation. Details are omitted to improve presentation clarity.

The spectral representation of a field at a given vertical layer (denoted by *Spectral(spec)*) is a summation of terms derived from the gaussian representation (denoted by *Component1(spec)* and *Component2(spec)*). These two terms change with latitude, that is, a second dimension (*lat*) was dropped to reduce memory usage. The summation is performed over the latitudes. A code fragment typical of this computation is

```
do lat = 1, latMax
...
  do spec = 1, specMax
    Spectral(spec)=Spectral(spec)+
        Component1(spec) + Component2(spec)
  end do
end do
```

where both *Component* arrays are computed in the *lat* loop, prior to the *spec* loop.

The outermost loop (*lat*) was parallelized. The values of *Component* are the same, bit by bit, for all latitudes, in the

sequential or parallel executions, with any number of processors. A critical section on the update of *Spectral* is required to guarantee program correctness.

But, it does not guarantee the same results due to round off in the order of summations. Since the latitude loop is split over processors and the scheduling of iterations to processors is dynamic, there is no guarantee that two executions will update *Spectral* in the same latitude order. Even with a fixed number of processors.

Two other solutions were tested. Both require restoring the second dimension (*lat*) of the *Component* arrays, so that values of these arrays computed at one iteration of the *lat* loop were saved for later use. The idea is to split the *spec* summation, partially inside the parallel loop, partially outside the parallel loop. This summation could be performed in one of two ways:

1. By *partial sums*, that is, each processor sums the components over its own iterations. Final sum of partial sums is performed sequentially, outside the parallel loop.
2. Sequentially, by moving the summation loop outside the parallel loop.

The first solution does not guarantee reproducibility of results even with a fixed number of processors, due to dynamic scheduling of iterations. The second solution guarantees reproducibility with any number of processors, at the expense of reducing parallelism.

The second solution was implemented, using a single, dimensionally augmented *Component* array. Each processor adds the two *Components* in a single array, in parallel. After the execution of the parallel loop, the array is reduced by accumulating over latitudes into the spectral field.

But that is not the sequential execution order. It accumulates *Component1* into the spectral field *before* adding *Component2*. This tiny difference in execution order is the reason why all parallel executions produce identical results, but they differ from sequential execution results.

B. Performance

Table II exhibits measured speed ups, computed by the ratio of the wall clock time of the sequential execution of the original code by the wall clock time of the parallel execution. It also shows the Mflops rate of each execution.

TABLE II
Measured SpeedUps

Processors	*Speed-up*	*MFlops*
1 (seq)	1	678
1 (par)	0.99	674
2	1.79	1216
3	2.48	1685
4	3.09	2096

Measured speed ups compare favorably with expected speed ups (see Table I). And, as a performance reference, it is worth mentioning that each NEC SX4 processor has a nominal top speed of 2000 Mflops. Consequently, the top speed of the machine is 8000 Mflops.

VII. Conclusions And Future Work

The feasibility of a massively parallel production weather forecast code was demonstrated around 1995. Speed-ups on the order of 900 over 1000 Cray T3D processors were obtained by that time [BAR 95]. These astonishing efficiency results were reproduced at many Weather Centers around the world [TRE 98], [EST 98]. Current international research focus is on parallelizing the complete forecast cycle, including new variational data assimilation techniques [KAU 98].

But parallelizing a production weather code is no easy task. The enterprise takes about five years of a considerable team of experts. It requires a complete restructuring of the code, including new data structures [BAR 95]. The major reason is that the original code was not designed for parallel execution.

The results reported in this work demonstrate what is achievable by a modest, time constrained effort. They encourage us in pursuing higher efficiency. Future work includes new data structures for the gaussian to spectral and spectral to gaussian transformations, including a new formulation designed for higher efficiency, at the cost of increasing memory usage. It is a sample of the work to be performed in the following years: change design goals of the forecast program from low memory usage to high parallel efficiency.

Acknowledgments

We gratefully thank Benicio P. de Carvalho Filho for the encouragement to write this work, as well as Celso L. Mendes for comments and suggestions on early versions of this paper. We also thank Atsuko Ohyama, Tadashi Murase and Nanaumi Nagamine from NEC SMD for their precious technical support.

References

[BAR 95] Barros, S. R. M. et al. The IFS model: A parallel production weather code. *Parallel Computing*, V.21, 1621-1638, 1995.

[BON 96] Bonatti, J. P. Modelo de Circulação Geral Atmosférica do CPTEC. *Climanalise, Edição Especial de 10 anos,* Oct 1996. Also available at http://www.cptec.inpe.br/*products/climanalise/cliespl 0a/bonatti.html*.

[EST 98] Estrade, J. F. et al. Operational Parallel Processing at Meteo-France. Eighth ECMWF Workshop in the use of parallel processors in meteorology – Towards

TeraComputing, Reading, England, Nov 1998. *To be published.*

[KAU 98] Kauranne, T. Parallel minimization for 4D Variational Data Assimilation, Karlman filtering and generating initial perturbations to Ensemble Forecasting. Eighth ECMWF Workshop in the use of parallel processors in meteorology – Towards TeraComputing, Reading, England, Nov 1998. *To be published.*

[KIN 97] Kinter, J. L. et al. The COLA Atmospheric Biosphere General Circulation Model. Technical Report #51, Center for Land and Atmosphere Studies, Maryland, USA, Oct 1997.

[NEC 96] NEC. Analyzer-P/SX Reference Manual. Part Number G1AF15E4. NEC Corporation, 1996.

[NEC 97] NEC. Fortran 77/SX Multitasking User's Guide. Part Number G1AF12E5. NEC Corporation, 1997.

[TRE 98] Trémolet, Y. and Sela, J. The parallel version of the NCEP global model. Eighth ECMWF Workshop in the use of parallel processors in meteorology – Towards TeraComputing, Reading, England, Nov 1998. *To be published.*